Universidad Zaragoza

# Trabajo Fin de Grado

## Sistema de captura de movimiento mediante sensores de posición para la visualización y valoración de ejercicios de rehabilitación

## Movement capture system using position sensors for the visualization and evaluation of rehabilitation exercises


Autor

**Eduardo Ejarque Pascual**

Director

**José Ramón Beltrán Blázquez**

Escuela de Ingeniería y Arquitectura

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación



2018

*A mi familia.*
*Por su amor y apoyo incondicional.*

*Veni Vidi Vici.*

# RESUMEN

Desde hace unos años ha comenzado la necesidad y utilidad de ser capaces de localizar a personas o elementos en el interior de un edificio, donde los Sistemas de Posicionamiento Global (GPS) son ineficaces en estos entornos.

En más profundidad, si no queremos sólo saber su posición, si no también poder capturar en tiempo real el movimiento que se está realizando, surge la idea de implementar un sistema de bajo coste que se pueda utilizar en el ámbito del e-Health o medicina inteligente.

Los múltiples ejercicios de rehabilitación existentes van relacionados con el tipo de lesión, donde la presencia de un profesional es fundamental para realizar el movimiento correcto. Ahora bien, existen movimientos sencillos, realizados tras una leve lesión o al llegar a la fase final de una rehabilitación más compleja, que se pueden realizar de forma autónoma.

En muchas ocasiones, son los propios pacientes o profesionales los que se dirigen a la consulta o domicilio para realizar este seguimiento y realización de la rehabilitación. En cualquiera de estos casos, existe un desplazamiento, donde si se trata de una persona mayor o estás situado en un núcleo rural, es bastante dificultoso.

Por este motivo, nace la idea de implementar un hardware de bajo coste, el cual posea un software para realizar una representación y valoración de ejercicios de rehabilitación. De esta manera el experto puede realizar un seguimiento a distancia y será capaz de aumentar el número de pacientes a su cargo.

# Tabla de contenido

# Capítulo 1. Introducción y objetivos

El presente proyecto se ha desarrollado dentro del marco de Trabajo Fin de Grado para optar al título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación en la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza, durante el período de junio a septiembre del año 2018.

La finalidad es implementar un sistema de detección de movimiento de bajo coste que esté enfocado a la valoración en tiempo real de ejercicios de rehabilitación.

Para la medición de los datos se va a utilizar un dispositivo hardware compuesto por un sensor de nueve grados de libertad que dispone de acelerómetro, giroscopio y brújula.

El objetivo planteado es realizar un *raw data analysis* (análisis de los datos en bruto) óptimo de todas las mediciones que realiza el sensor, para obtener una información adecuada, eliminando al máximo los errores que introduce el sistema y así, posteriormente, traducirlos en un movimiento suave y preciso en un entorno gráfico 3D.

El sensor estará conectado a un microcontrolador (circuito integrado programable) Arduino Nano, donde se realizará el filtrado de los datos y el cálculo de la posición. Otro objetivo es utilizar la máxima capacidad computacional del microcontrolador para enviar por el puerto serie los datos y ser representados directamente.

Tras este primer análisis, el microcontrolador, gracias a un módulo bluetooth previamente configurado, se comunica inalámbricamente con el dispositivo donde estamos ejecutando la aplicación que representará el movimiento. El software para el desarrollo de la representación será Unity, donde utilizamos una programación orientada a objetos basada en lenguaje C-Sharp y el software Visual Studio para su compilación.

Así, el sistema a implementar se basa en un conjunto compuesto por sensor inercial, módulo bluetooth, microcontrolador y software de representación.

La presente memoria se compone de siete capítulos. El primero realiza una introducción al problema y establece los objetivos del proyecto. Seguidamente se analizan los sistemas de posicionamiento y detección de movimiento para desarrollar los requisitos del trabajo.

En el tercer capítulo se indaga en el desarrollo hardware, explicando uno a uno los componentes, su comunicación y conexionado.

Posteriormente se realizará el análisis de los datos en el capítulo cuatro, para exponer como se ha realizado el entorno de representación en el capítulo cinco. Para finalizar, en el capítulo seis se expondrán las pruebas realizadas para concluir en el último capítulo con las conclusiones y visión futura de este trabajo.

# Capítulo 2. Análisis y diseño del sistema

## 1. Análisis del problema

El propósito del sistema es hallar, a partir de los datos del acelerómetro en los tres ejes, la posición del cuerpo. Si reducimos el ámbito a un eje, el procedimiento para extraer la posición es realizando una doble integración de la aceleración:

$$\iint_0^t a(t)dt\,dt$$

donde a(t) es la aceleración.

Para resolver la anterior integral, es necesario aplicar la integración numérica, la cual será explicada en el siguiente apartado y pondremos en práctica más adelante.

En la resolución de estas operaciones, debemos tener claro el sistema de coordenadas el el que se va a trabajar.

Tras esto, realizaremos un recorrido histórico explicando cómo han evolucionado los sistemas de posicionamiento y captura de movimiento, para poner en contexto el objetivo final y escoger el sistema que se pondrá en práctica.

## 2. Integración numérica

La integración numérica se compone de una gran variedad de algoritmos para calcular el valor numérico de una integral definida. Además, se utiliza para resolver ecuaciones diferenciales.

El problema más sencillo considerado por la integración numérica es el de calcular una solución aproximada a la integral definida:

$$g(a,b) = \int_a^b f(x)dx$$

Las razones para llevar a cabo la integración numérica son principalmente la imposibilidad de realizar dicha integración de forma analítica. Por lo que integrales que requerirían de un gran conocimiento y manejo de matemática avanzada pueden ser resueltas de forma más sencilla mediante estos métodos numéricos que nos proporciona la integración numérica [1].

## 3. Sistemas de coordenadas

Los sistemas de coordenadas fundamentales que se utilizan son los ortogonales, dextrógiros y cartesianos. Se diferencian en el origen al que se referencian, la orientación relativa de sus ejes y el movimiento relativo entre sus planos. En función del movimiento que se desea estudiar, es mejor utilizar unos sistemas que otros.

Para el sistema que se implementa, es importante estudiar de una forma adecuada la orientación en la cual se está realizando el movimiento. Es entonces cuando aparece un método para describir este parámetro.

### 3.1. Ángulos de Euler

Se trata de un método muy utilizado para especificar la orientación angular de un sistema de coordenadas con respecto a otro. Se basa en la utilización de tres ángulos, los cuales, mediante una sucesión ordenada de giros, definen el cambio de un sistema a otro.

Los ángulos de Euler son llamados con el nombre de $roll\ (\phi)$, $pitch\ (\theta)$, $yaw\ (\psi)$, convencionalmente utilizados en navegación para calcular la actitud de un móvil [2].

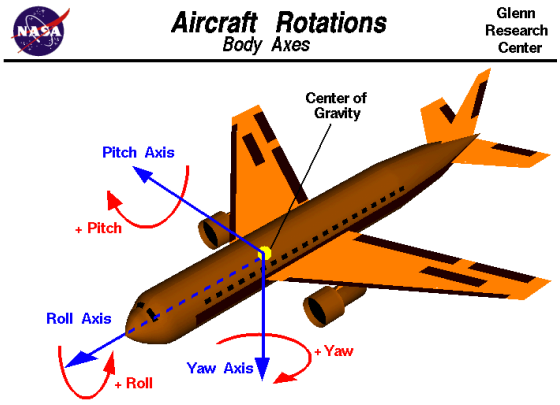



*Figura 1. Representación de los ángulos de pitch, roll y yaw. Fuente: www.grc.nasa.gov*

Observando la *Figura 1*, el ángulo de *roll* es la rotación del eje x, es decir, el giro de las alas del avión. El ángulo de *pitch* mide la rotación sobre el eje y, por tanto, se traduce en la inclinación del morro del avión. Por último, el ángulo *yaw* mide la rotación del eje z, que es el giro del morro del avión con respecto al norte.

Cabe destacar que los ángulos de Euler no están unívocamente definidos y existen ambigüedades. Por este motivo, las rotaciones deben realizarse siempre en el mismo sentido, ya que, si los ángulos son aplicados en otro orden, dan lugar a transformaciones diferentes.

# 4. Sistemas de posicionamiento

## 4.1. Recorrido histórico

Hace tan sólo 30 años, para viajar se utilizaba un mapa a papel, donde cualquier cálculo se realizaba con imprecisión. A finales de los 90 aparecieron páginas web que calculaban rutas ofreciendo distancias e indicaciones más o menos precisas, pero sin fiabilidad.

Con el nuevo siglo se comenzaron a utilizar navegadores por satélite, capaces de dar una posición casi exacta e incluso recalcular rutas. Actualmente los dispositivos móviles incluyen esta tecnología y pueden guiarnos tanto por montaña como por núcleos urbanos.

Ahora bien, quedan zonas que se resisten, se trata de los espacios interiores, ambientes donde se desarrolla el sistema. Es aquí donde los sistemas por satélite llegan con una señal débil, que provoca metros de inexactitud, inadmisible en una captura de movimientos pequeños.

## 4.2. Técnicas de posicionamiento en interiores

Estos sistemas tan recientes han experimentado una gran evolución en los últimos años. A continuación, basándonos en la descripción que realiza Cristina Regueiro en su trabajo de investigación *"Error en el posicionamiento indoor en dispositivos móviles"* se detallan los principales métodos para el posicionamiento en interiores.

### 4.2.1. Marcadores fijos

Es la primera técnica que apareció, donde la infraestructura se compone de marcadores situados en diferentes puntos del entorno que son reconocibles por un dispositivo específico.

Las principales ventajas de estos sistemas son su sencillez y bajo coste de despliegue. Además, al tener las marcas localizadas, se consigue una gran precisión en la localización. Sin embargo, no son sistemas de localización propiamente dichos, ya que la localización se produce por la acción del usuario tras reconocer el marcador, lo que limita al usuario a estar localizado únicamente en los marcadores. La existencia de un obstáculo que no haga visible el marcador es un gran problema, pues no será posible la localización [3].

### 4.2.2. Sistemas inalámbricos

Las tecnologías inalámbricas se clasifican en función de la frecuencia que poseen sus ondas, como pueden ser infrarrojos, radiofrecuencia (RFID, WIFI, Bluetooth…) y ultrasonidos.

Se basa en el uso de ondas electromagnéticas para obtener la localización del usuario con respecto a un punto de referencia conocido. Tiene la ventaja de que se puede implementar fácilmente, siempre que exista una infraestructura de red previa, principal inconveniente.

Dado que en la actualidad la mayoría de edificios cuentan con infraestructura de red WIFI, sumado a que los teléfonos móviles cuentan con receptores de WIFI o *Bluetooth*, este sistema es el más estudiado, siendo fácil de desplegar y con una rápida extensión. No podemos olvidar que siempre existirán zonas donde la señal no llegue de forma adecuada [3].

### 4.2.3. Sistemas inerciales

Ante la necesidad de dependencia de una infraestructura de red que necesita el sistema anterior, surgen nuevos sistemas autónomos que requieren un único hardware, sin necesidad de receptores. Estos elementos serán sensores que llevarán incorporados acelerómetro, giroscopio y brújula, siendo capaces de determinar la aceleración u orientación de un movimiento.

Por tanto, la gran ventaja de este sistema es que no necesita una infraestructura externa, lo que hace que el coste de su despliegue sea mínimo. Añadir que en la actualidad muchos dispositivos cuentan con sensores inerciales, por lo que su implementación sería rápida.

Sus inconvenientes principales son, en primer lugar, la necesidad de una calibración inicial de gran exactitud. Por otro lado, el error que introducen estos dispositivos es acumulativo y puede ser desastroso en el tiempo. Por este motivo es necesario realizar calibraciones cada cierto tiempo [3].

Como conclusión, en base a las características de los sistemas mencionados, podemos indicar que un sistema de posicionamiento inalámbrico, combinado en las zonas de baja señal con un sistema inercial, llevaría a una solución muy adecuada.

## 5. Sistemas de captura de movimiento

### 5.1. Recorrido histórico

Los inicios de la captura de movimiento se iniciaron a finales del siglo XIX, donde Eadweard Muybridge colocó una serie de cámaras que se activaban mediante hilos.

Un anuncio publicitario en la década de los 80 se convirtió en el primer documento audiovisual en recurrir a la captura del movimiento. El programador Robert Abel realizó, mediante una serie de puntos negros en las articulaciones de una actriz, un robot 3D [4]. El método fue una sesión de fotos en la que se realizaron instantáneas desde diferentes ángulos. Posteriormente todas las capturas se pasaron por un software que recreó la figura.

El gran salto de esta tecnología se vio en el personaje de cine *´Gollum`,* que parecía tener vida. Actualmente en la industria del videojuego los personajes son actores reales.

A continuación, se realizará una breve explicación de cada uno de los sistemas de captura de movimiento que existen en la actualidad, para justificar la elección del sistema utilizado en el proyecto.

### 5.2. Sistema óptico

Es el más utilizado en la industria cinematográfica. Posee la base del ya mencionado robot de Robert Abel: situar puntos estratégicamente en el cuerpo del actor para posteriormente recoger los datos en un software. Esta técnica se divide en pasiva y activa.

En el sistema de óptica activa, cada punto que está situado sobre el actor emite luz de manera autónoma. Es el traje el que facilita la "imagen" a la cámara, que simplemente recoge lo que le llega a través de esta luz proyectada. En la óptica pasiva, los marcadores aparecen apagados y es la cámara (infrarroja) la que se encarga de emitir el destello para que sea reflejado en el actor y posteriormente registrarlo.

Aunque ambos sistemas tienen sus complicaciones, actualmente la industria utiliza la óptica pasiva. El primer motivo es ergonómico, donde los actores pueden moverse con total movilidad al no existir cableado. Esta total libertad puede producir en ocasiones errores en el rebote, donde el software es capaz de interpretar en qué puntos se situaría el cuerpo [5].

### 5.3. Sistema electromecánico y mecánico

El sistema electromecánico apuesta por la utilización de sensores magnéticos con multitud de cables y con una alta posibilidad de perder la señal.

En el sistema mecánico se utiliza un exoesqueleto que va adherido al actor, el cual proporciona poca ergonomía pero tiene a favor que no se necesita un procesamiento tras el movimiento [5].

### 5.4. Sistema inercial

Con la misma base que la explicada para los sistemas de posicionamiento, actualmente se han introducido sensores inerciales (IMU – Inertial Measurement Unit) para este tipo de aplicaciones. Su punto fuerte es que proporcionan en tiempo real la orientación del lugar donde son ubicados. Por el contrario, necesitan de una calibración continua y muy precisa [5].

## 6. Análisis de requisitos

Como podemos observar en base a las características de los sistemas anteriores, para el cálculo de la posición descartamos la utilización de puntos fijos y el sistema inalámbrico debido a la infraestructura que sería necesaria desplegar.

En la captura de movimiento a gran nivel, el proceso se realiza en dos fases: la primera, de grabación del movimiento y la segunda realizar un procesado de los mismos para hacer la recreación en tres dimensiones. Estos sistemas requieren de un gran coste económico tanto de hardware como de software.

Por tanto, concluimos que la utilización de sensores inerciales para determinar el movimiento y posición es una solución muy adecuada, haciendo al sistema autónomo, sin depender de infraestructura externa. Poseerá los siguientes requisitos:

- Sistema compuesto de sensores inerciales
- Conectividad inalámbrica durante todo el ejercicio
- Comunicación I2C entre sensor y microcontrolador
- Autonomía de alimentación del dispositivo para varias sesiones
- Precisión y suavidad en la representación
- Avisos adecuados cuando hay un mal gesto
- Garantizar una relativa ergonomía del dispositivo
- Hardware de bajo coste

Una vez conocido el funcionamiento y requisitos, podemos proceder al modelado, diseño e implementación del sistema.

## 7. Modelado del sistema

El sistema a implementar en este proyecto posee la siguiente estructura:

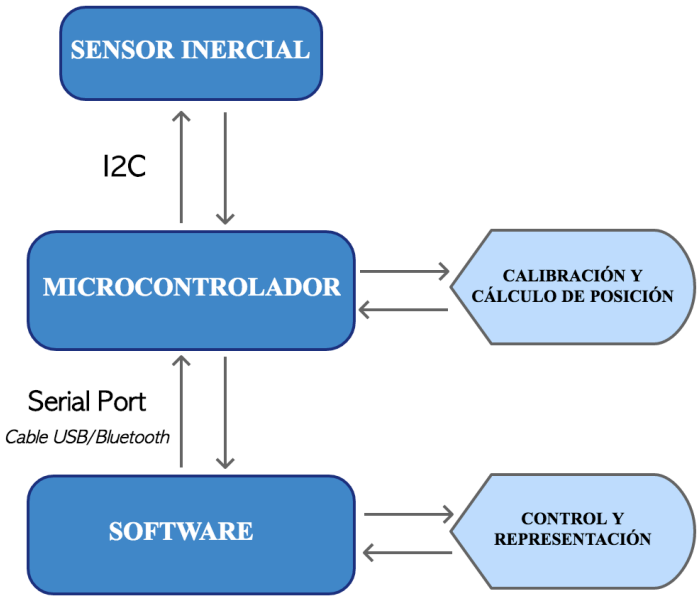



*Figura 2. Modelado general del sistema*

# Capítulo 3. Desarrollo hardware

## 1. Introducción al capítulo

En este capítulo se realiza una ilustración de los elementos utilizados, así como una explicación de la comunicación entre cada uno de ellos. Primeramente, analizaremos la base de funcionamiento de cada uno de los elementos de los que se compone un sistema inercial y cómo se realizan las mediciones. Posteriormente, analizaremos en profundidad las características del sensor, microcontrolador y módulo bluetooth escogidos.

Los elementos principales son:

- Sensor Adafruit LSM9DS0 (acelerómetro + giroscopio + brújula)
- Microcontrolador Arduino Nano
- Módulo Bluetooth HC-06

Tras esto, se analizará cómo están conectados cada uno de los dispositivos empleados, así como el puerto, protocolo y configuraciones utilizadas para tener una correcta comunicación entre todos los elementos.

## 2. Descripción de los elementos

### 2.1. Componentes de un sistema inercial

#### 2.1.1. Acelerómetro

Un acelerómetro es un elemento destinado a medir aceleraciones. No tiene por qué verse desde el punto de vista de la aceleración como la variación que experimenta la velocidad en el espacio. Esta aceleración medida se asocia a las fuerzas que experimenta una masa de prueba, utilizando la segunda Ley de Newton $(F = m.a)$. En el capítulo cuatro se explica en profundidad las componentes cinemáticas empleadas.

Los acelerómetros más utilizados son el mecánico y el piezoeléctrico.

El acelerómetro mecánico es el más simple. Se construye uniendo una masa a un dinamómetro [6] cuyo eje está en la misma dirección que la aceleración que se desea medir.

El piezoeléctrico (Figura 3) se basa en un elemento retículo cristalino, el cual está comprimido por una masa. Cuando se comprime el elemento piezoeléctrico, se produce una carga eléctrica proporcional a la fuerza aplicada, por lo que cuando el conjunto es sometido a vibración, el disco piezoeléctrico se ve sometido a una fuerza variable, proporcional a la aceleración de la masa.

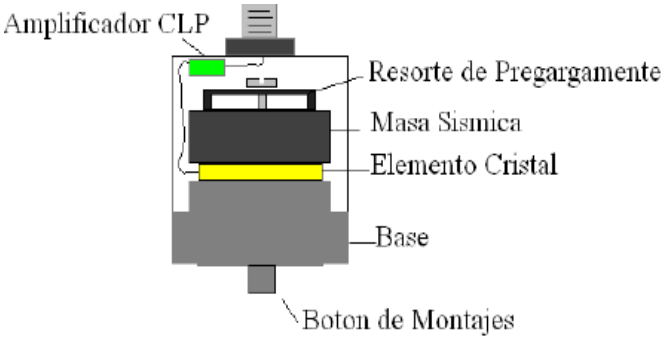



*Figura 3. Estructura de un acelerómetro piezo-eléctrico*

Por su mayor precisión, se utilizará un acelerómetro piezoeléctrico.

### 2.1.2. Giroscopio

Un giroscopio es un dispositivo de medida de la orientación, basado en los principios de conservación del momento angular.

Consiste en un cuerpo con simetría de rotación que gira alrededor del eje de dicha simetría. Cuando se aplica una fuerza, su eje cambia de orientación (experimenta un momento angular) y éste a su vez gira sobre un tercer eje, perpendicular tanto a aquel respecto del cual se lo ha empujado a girar, como a su eje de rotación inicial [7].

Convencionalmente suelen utilizarse giroscopios mecánicos, por lo que nuestro sensor poseerá este tipo de elemento, el cual nos proporcionará la velocidad angular.

De esta manera, el giroscopio junto con el acelerómetro, van a formar un sistema de seis grados de libertad (Figura 3):

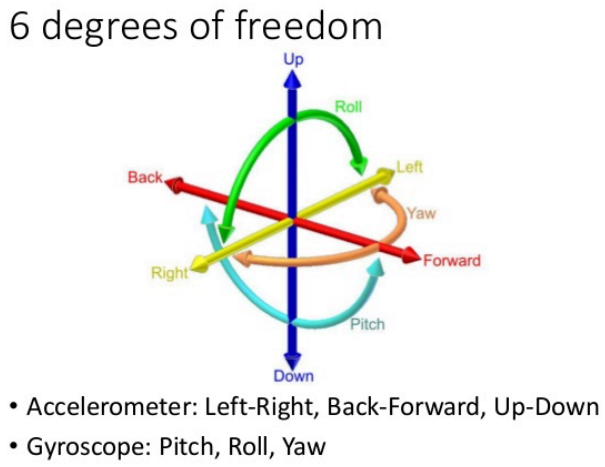



*Figura 4. Sistema con seis ejes de libertad. Fuente: Wikipedia. Autor: Horia Ionescu.*

### 2.1.3. Magnetómetro

Más conocido con el nombre de brújula, es un instrumento para medir la fuerza y dirección del campo magnético en el área cercano al dispositivo.

En los dispositivos móviles, en general, se emplean magnetómetros vectoriales, que tienen la capacidad de medir la componente de campo magnético en una determinada dirección relativa a la orientación espacial del propio dispositivo [8].

Es por esta razón por la que en nuestro sensor el tipo de brújula será vectorial.

## 2.2. El sensor Adafruit LSM9DS0

Este dispositivo se trata de un sensor inercial que combina acelerómetro piezoeléctrico, giroscopio mecánico y brújula vectorial, todos con tres ejes de referencia.

Los rangos de escala del acelerómetro son ± 2 *g*, ± 4 *g*, ± 6 *g*, ± 8 *g*, ± 16 *g*, los del giroscopio son de ± 245, ± 500, ± 2000 °/s (dps) y de la brújula ± 2 *g*, ± 4 *g*, ± 8 *g*, ± 12 *g.*

Sus características principales se pueden resumir:

- Voltaje de entrada de entre 3.3-5V
- Acelerómetro, giroscopio y brújula programables en la escala necesaria
- Incluye un sensor de temperatura digital que opera entre -40-85ºC
- Conexión directa mediante I2C a través de los pines 3V, GND, SCL y SDA
- Tamaño muy reducido (0,65x0,33cm)

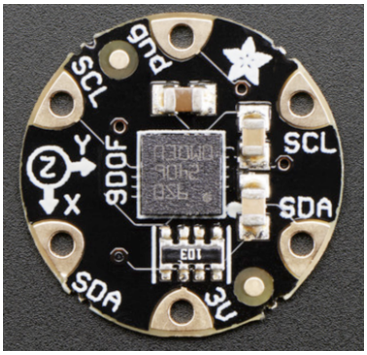



*Figura 5. Sensor Adafruit LSM9DS0*

Para una información más detallada de este sensor consultar *datasheet* en anexo 1.

## 2.3. Placa de Microcontrolador Arduino Nano 3.0

Este pequeño microcontrolador de código abierto (*open-source* [9]) posee grandes características que lo hacen una buena opción para utilizar en un sistema de tamaño reducido.

- Alimentación: vía conexión mini-USB o en el pin 27 (éste siempre que esté regulada a 5V) entre otras opciones.
- Memoria: el procesador ATmega320 le proporciona una *Flash Memory* de 32KB y una *SRAM* de 2KB
- Entradas/Salidas digitales: posee 14 pines digitales que operan a 5V, muchos de los cuales tienen funcionalidades predeterminadas, aunque pueden actuar tanto de entrada como de salida programados adecuadamente.
- Entradas/Salidas analógicas: posee 8 entradas analógicas referenciadas a 5V, con una resolución de 10 bits
- La velocidad de su reloj es de 16MHz
- Posee un botón de *reset* incorporado

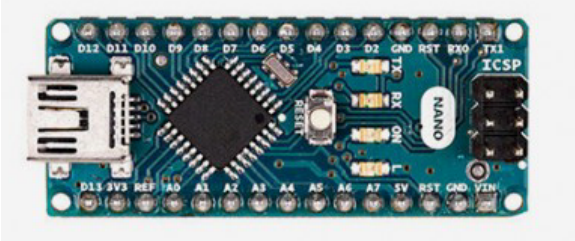



*Figura 6. Microcontrolador Arduino Nano*

Para una información más detallada de este sensor consultar *datasheet en* anexo 2.

## 2.4. Módulo Bluetooth BLE-HC-06

- Puede a operar con bajo voltaje (3.1 - 4.2V)
- Transmisión y recepción a 2.4GHz con antena incluida
- Bluetooth 2.0 de bajo consumo
- Tasa de error de bit cercana a cero
- Fácil configuración

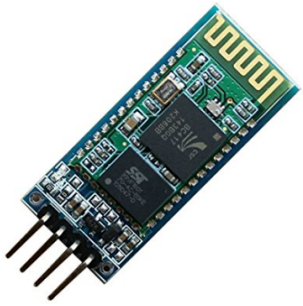


*Figura 7. Módulo Bluetooth HC-06*

Para una información más detallada de este sensor consultar *datasheet* en anexo 3.

### 2.5. Lista de materiales

En la siguiente tabla se detalla el coste de cada dispositivo empleado para la implementación del hardware.

| Elemento | Cantidad | Coste/unidad |
|---|---|---|
| Microcontrolador Arduino Nano 3.0 | 1 | 35,21€ |
| Adafruit LSM9DS0 | 1 | 20,85€ |
| BLE Module Bluetooth HC-06 | 1 | 11,77€ |
| Pila Alcalina 9V Duracell | 1 | 2,41€ |
| **Total** | **4** | **70,24€** |

*Tabla 1. Lista de materiales y precios del sistema hardware implementado*

## 3. Conexiones

### 3.1. Conexión sensor – microcontrolador

El protocolo de comunicación elegido (en el siguiente punto explicaremos la comunicación y el porqué del protocolo elegido) nos marca la conexión a seguir entre el dispositivo inercial y el microcontrolador.

Serán necesarias cuatro conexiones, desde nuestro sensor:

1) $V_{in}$(*cable rojo*) al pin de 3.3V de nuestro Arduino Nano
2) GND (*cable negro*) a cualquier punto con referencia a masa del Arduino
3) SDA (*cable verde*) al pin A4
4) SCL (*cable azul*) al pin A5 (Ver Figura 8)

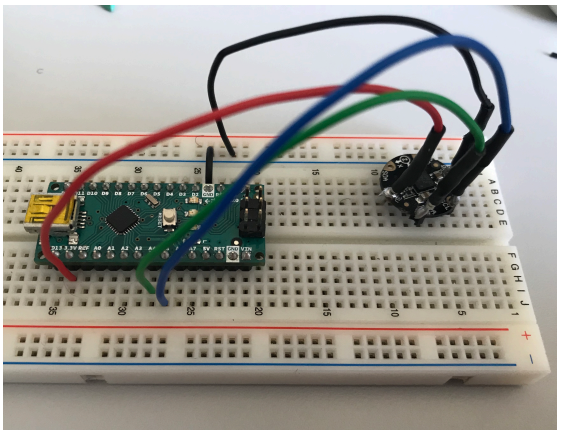


*Figura 8. Conexión sensor - microcontrolador*

## 3.2. Conexión microcontrolador – PC

La comunicación entre el microcontrolador y el PC mediante el puerto serie la podemos realizar de dos formas, la primera con una conexión USB y la otra de modo inalámbrico a través de un módulo bluetooth.

### 3.2.1. Conexión USB

Como se ha mencionado antes, la placa de Arduino Nano posee una entrada de *miniUSB*. Para la conexión se necesita un cable estándar de *USB tipo B* a *miniUSB* (ver Figura 9).

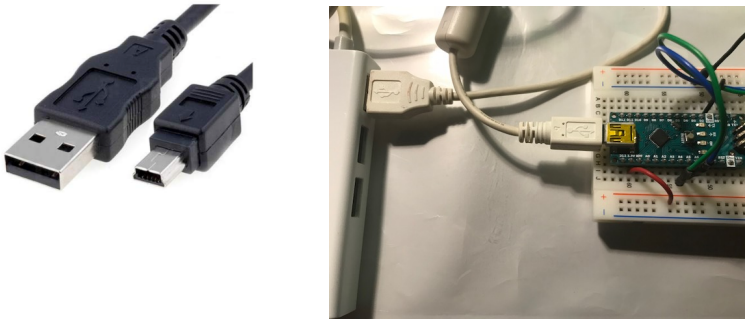


*Figura 9. Cable mini USB – USB-B (izquierda). Conexión microcontrolador-PC (derecha)*

### 3.2.2. Conexión inalámbrica

Existen diferentes formas de realizar la conexión sin cables a nuestro PC y de todas ellas hemos elegido la comunicación bluetooth, debido a su gran conectividad en distancias cortas (inferiores a 10 metros), bajo consumo de batería y un coste económico del módulo hardware bajo.

Para su conexión, en el microcontrolador definimos el puerto Bluetooth ayudándonos de la librería *SoftwareSerial.h*, donde recibe como argumentos los pines de transmisión y recepción.

El siguiente ejemplo se traduce en que la pata de transmisión del módulo BT estará conectada al pin 2 de Arduino *(cable verde),* y la de recepción al pin 3 del microcontrolador *(cable azul).* De esta forma tenemos $TRX_{arduino} - RCX_{bluetooth}$ y viceversa. Por último, acordarnos de llevar al punto común de masa el pin GND del módulo y al pin de 3.3V del arduino la alimentación $V_{cc}$ (ver Figura 10).

```
SoftwareSerial BTSerial (2,3); //2 RX, 3 TX.
```

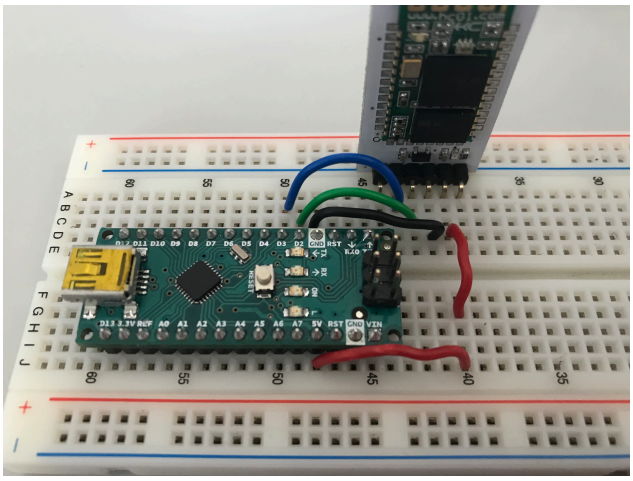


*Figura 10. Conexión del módulo Bluetooth al microcontrolador*

### 3.3. Conexionado de todo el sistema

En la figura 11 se puede ver el conexionado del sistema hardware completo necesario para la realización del proyecto.

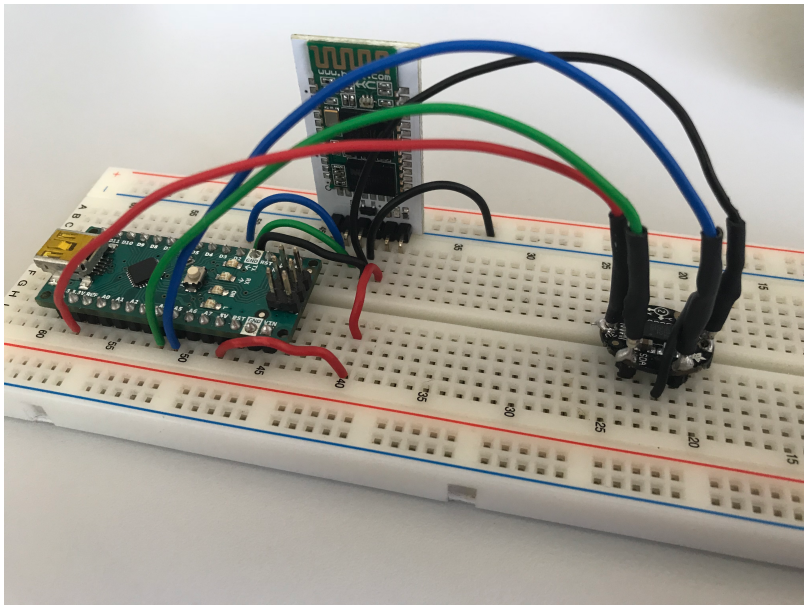


*Figura 11. Sistema hardware implementado*

# 4. Comunicaciones

## 4.1. Protocolos de comunicación: UART vs SPI vs I2C

Por su funcionamiento asíncrono y baja velocidad de transmisión de los datos, comenzamos descartando el protocolo UART (*Universal Asynchronous Receiver/Transmitter*) para nuestro sistema [10].

Ahora tenemos que elegir entre dos protocolos síncronos con estructura de maestro-esclavo (*master-slave* [11]) que poseen buena tasa de transmisión.

Respecto al número de conexiones necesarias, SPI necesita como mínimo cuatro conexiones para comunicar dos dispositivos, una de ellas para el reloj serie (SCLK), una para la salida del *master*/entrada al *slave* (MOSI), otra para la salida del *slave*/entrada al *master* (MISO) y una cuarta conexión (SS) para seleccionar el dispositivo. A partir de esta estructura, por cada nuevo elemento en nuestro sistema será necesario realizar una nueva conexión.

Sin embargo, I2C necesita únicamente dos conexiones, la de datos (SDA) y la de reloj (SCL) que crea el *master* para controlar el envío de datos. La gran ventaja de I2C es que realiza un direccionamiento único de 7 bits para cada nuevo dispositivo que se añade (el fabricante preestablece el valor de esta dirección).

Como conclusión, SPI tiene la ventaja de ser *full-duplex* a costa de utilizar más conexiones, donde un sólo elemento es siempre el *master.* En cambio, I2C posee comunicación *half-duplex* con menos conexiones y gracias al direccionamiento podemos cambiar la funcionalidad de cada dispositivo, por lo que podemos tener varios *master en nuestro sistema.*

Tras este análisis, hemos optado por elegir la conexión mediante **I2C**, debido a su fácil conexión, disponer de soporte por hardware en nuestro microcontrolador (Arduino posee pines vinculados específicamente a comunicación por I2C) y un sistema de control de flujo.

Además, nuestro sensor LSM9DS0 viene con interfaz de bus I2C integrada que soporta velocidades entre 100kHz y 400kHz.

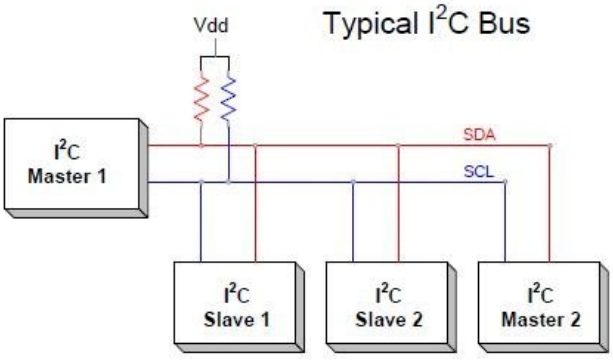



*Figura 12. Bus de comunicación I2C [12]*

## 4.2. Comunicación vía puerto serie

Para realizar la transmisión y recepción de datos, ya sea mediante cable o de forma inalámbrica, se utilizará el **puerto serie** de nuestro Arduino Nano.

Un dato muy relevante es que la comunicación mediante puertos serie es **asíncrona,** al contrario que el protocolo utilizado (I2C) por lo que no se transmiten datos asociados a un reloj para sincronizar el envío y recepción.

La solución que establece el puerto serie para sincronizar la transmisión y recepción es típica en una transmisión asíncrona, realizar el envío mediante trenes de pulsos de un byte, donde se utilizan el primer y último bit para la sincronización, los llamados **bit de *start*** y **bit de *stop*** (ver Figura 12).

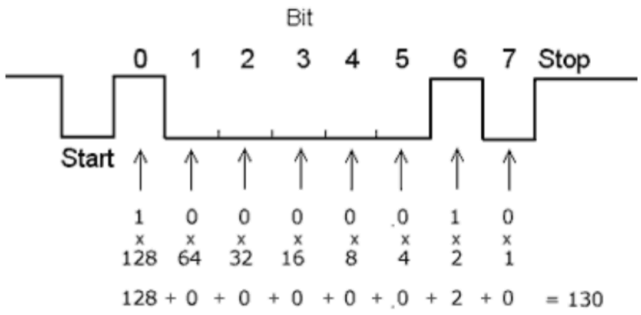



*Figura 13. Estructura de trama en el puerto serie. Fuente: bibliografía número nueve.*

## 4.3. Configuración Arduino

Para trabajar con el sensor elegido, es necesario importar las librerías que nos proporciona el fabricante. Estas librerías son *Adafruit_Sensor.h* y *Adafruit_LSM9DS0.h.*

Si queremos habilitar el bus I2C de Arduino, debemos apoyamos en la librería *Wire.h*, la cual contiene las funciones necesarias para controlar el hardware integrado [13].

Como vemos en el siguiente ejemplo, en el cual estamos declarando nuestro sensor, al estar utilizando I2C como protocolo de comunicación es necesario asignar una única dirección para el bus de datos.

```
Adafruit_LSM9DS0 lsm = Adafruit_LSM9DS0(1000);// Use I2C,ID #1000
```

Tras establecer el protocolo de comunicación y su dirección, lo primero que hacemos es abrir el puerto serie, indicando la velocidad de transmisión a la que queremos trabajar. Es muy importante que todos los dispositivos que interfieren en la comunicación estén a la misma velocidad.

```
Serial.begin(9600); //En el caso de utilizar cable USB
BTSerial.begin(9600); //Para módulo bluetooth
```

Volviendo a lo mencionado en el apartado anterior, como utilizamos un cable USB el cual también posee comunicación por puerto serie, la recepción de 8 pulsos en el microcontrolador afecta a la manera en la que leemos por el puerto serie, ya que en nuestro sistema debemos diferenciar la recepción de un *flag* para, en función de un valor u otro, calibrar el dispositivo o comenzar el cálculo de posiciones

.

Por este motivo, lo que hacemos es leer por el puerto serie como si de una cadena de caracteres se tratara, para luego realizar la transformación (*parseado*) a entero.

```
char *cadena;
char data = Serial.read();
*cadena = data;
int flag = atoi(k);
```

Por último, obtenemos los datos en cada eje acordes a como nos indica la librería del sensor que estamos utilizando y posteriormente los enviamos por el puerto serie vía USB o Bluetooth mediante la función *Serial.print()*

```
sensors_event_t accel, mag, gyro, temp;
lsm.getEvent(&accel, &mag, &gyro, &temp);
```

*Comunicación por USB:*

```
Serial.print(accel.acceleration.x); Serial.print(“,”);
Serial.print(accel.acceleration.y); Serial.print(“,”);
Serial.print(accel.acceleration.z); Serial.print(“,”);
```

*Comunicación por bluetooth:*

```
BTSerial.print(accel.acceleration.x); Serial.print(“,”);
BTSerial.print(accel.acceleration.y); Serial.print(“,”);
BTSerial.print(accel.acceleration.z); Serial.print(“,”);
```

Añadir que Arduino en su entorno de desarrollo nos proporciona dos herramientas para la visualización de los datos que transmitimos, la primera, llamada *Monitor Serie*, donde podemos ver los datos en forma numérica, y la segunda, *Serial Plotter,* que nos realiza una representación gráfica de los mismos. La única característica que no es posible es la visualización al mismo tiempo de las dos ventanas.

### 4.4. Configuración Unity

Como se ha indicado en los requisitos del sistema, el entorno de desarrollo elegido para la visualización de los datos es Unity, que nos permite una comunicación vía puerto serie realizando la siguiente estructura programada en lenguaje *C#* [14].

Para abrir un puerto serie debemos declarar una variable de tipo *SerialPort*, la cual necesita como parámetros un *string* que será el puerto del PC donde tenemos conectado nuestro cable USB *("/dev/tty.usbserial-A906P0GW")* o en el caso de ser comunicación bluetooth el puerto inalámbrico correspondiente *(“/dev/tty.HC-06-DevB”)* .

El segundo parámetro, de tipo *int*, será la velocidad de transmisión a la que trabajaremos. Es importante recordar lo anteriormente mencionado, que debe ser la misma en todos los dispositivos.

```
public string portName;
public SerialPort arduino;
arduino = new SerialPort(portName, 9600);
arduino.Open();
```

Para poder separar los datos aleatorios *(Raw Data)* y darles sentido, desde Arduino hemos enviado las componentes separadas por una coma. De esta manera, utilizando la función *ReadLine(),* la cual devuelve un *string,* podemos realizar, ayudándonos de la funcionalidad que nos da *Split,* la separación correcta de los datos para su posterior *parseado* al tipo de dato que queremos (*float*).

```
string dataString;
string[] dataBlocks;
float[] pos = new float[3]; //vector de posiciones recibido

dataString = arduino.ReadLine();
dataBlocks = dataString.Split(',');

pos[0] = float.Parse(dataBlocks[0]);
pos[1] = float.Parse(dataBlocks[1]);
pos[2] = float.Parse(dataBlocks[2]);
```

Asimismo, para enviar información por el puerto serie utilizaremos la función *Write*, poniendo entrecomillado el valor a enviar o directamente la variable.

```
arduino.Write("1");
```

## 4.5. Comunicación vía puerto serie mediante Bluetooth

### 4.5.1. Configuración Arduino

Para poder trabajar con el módulo bluetooth, primeramente, debemos importar la librería *SoftwareSerial.h* que nos permite crear el dispositivo, donde *BT* es el nombre de nuestro objeto de tipo *SoftwareSerial* y al que le pasamos como argumentos los pines que vamos a utilizar para recepción y transmisión.

Finalmente lo iniciamos utilizando la conocida función *.begin* pasándole como parámetro la velocidad (en baudios) de comunicación por el puerto serie.

```
SoftwareSerial BT(2,3); // RC | TX
BT.begin (9600);
```

Ya tenemos el dispositivo creado para comunicarse por el puerto serie, por tanto si queremos leer y escribir por éste, lo haremos utilizando las funciones *read()* y *write()*.

```
if(BT.available()){
  Serial.write(BT.read());
}

if(Serial.available()){
  BT.write(Serial.read());
}
```

De esta forma estamos enviando por el puerto serie lo que leemos por el bluetooth, pero también, al contrario, todo lo que enviamos por el terminal de Bluetooth se está volcando al puerto serie. No es exactamente lo que queremos realizar en el sistema, ya que estaríamos accediendo a la vez al puerto serie, tanto desde el módulo bluetooth como desde Unity.

La configuración es más sencilla, los datos sólo los tenemos que escribir por el puerto *BT*, donde accedemos desde Unity a la lectura y procesado de los datos.

Este será en formato de los datos a enviar, en esta ocasión utilizamos *print():*

```
if(Serial.available()) {
  BT.print(pos[0]); BT.print(“,”);
  BT.print(pos[1]); BT.print(“,”);
  BT.println(pos[2]);
}
```

### 4.5.2. Configuración Unity

Respecto a la configuración en el software Unity, recalcar que la comunicación sigue realizándose por el puerto serie.

Desde el microcontrolador escribimos los datos en puerto serie del bluetooth, y es éste el que los envía al puerto serie del dispositivo/PC, que está conectado por bluetooth al microcontrolador.

Por ello, sólo es necesario indicar correctamente el puerto adecuado.

# Capítulo 4. Tratamiento de los datos

## 1. Introducción

Tras la elección de los elementos hardware, así como su conexión y protocolo de comunicación óptimo, en este capítulo se tratan los métodos empleados para realizar el análisis de los datos, estudio de los errores y soluciones aplicadas.

## 2. Cálculo de la posición a partir de los datos del acelerómetro

### 2.1. Introducción

Como se ha mencionado con anterioridad, debemos utilizar la integración numérica para estimar la posición del objeto.

La aceleración es la variación de la velocidad por unidad de tiempo, es decir, la razón de cambio de la velocidad respecto al tiempo, matemáticamente:

$$a = \frac{dv}{dt} \qquad\qquad v = \frac{dp}{dt}$$

Así mismo, nuestro objetivo es a partir de la aceleración, conseguir una posición, por lo que realizaremos lo contrario de la derivada, una integración.

$$v = \int a(t) \qquad\qquad p = \int v(t)$$

Para ello, una técnica para su resolución es la **regla del trapecio**, uno de los métodos numéricos más utilizados para resolver integrales numéricas definidas.

### 2.2. Regla del trapecio

La regla del trapecio se basa en aproximar el valor de la integral de f(x) por el de la función lineal, la cual pasa por los puntos a y b. Por tanto, a partir de estos puntos establecemos que el área bajo la recta es una aproximación de la integral [15]:

$\int_a^b f(x)dx$

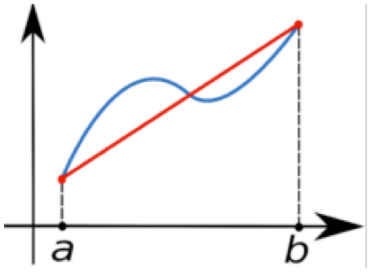



*Figura 14. La función f(x) (en azul) es aproximada por la función lineal (en rojo)*

Como resultado obtenemos lo siguiente (desarrollo completo en anexo 4):

$$\int_a^b f(x)dx \approx (b-a)\,\frac{f(a)+f(b)}{2}$$

Por tanto, en el siguiente punto aplicaremos el resultado alcanzado para calcular las ecuaciones necesarias en nuestro sistema.

## 2.3. Cálulo de la velocidad y posición

Dado la aceleración y el tiempo en el que se produce la lectura en un instante k, $aceleración_k$ y $tiempo_k$ respectivamente, la velocidad en ese instante, $velocidad_k$, puede expresarse como [16]:

$$velocidad_k = velocidad_{k-1} + 0.5\,.\,(tiempo_k - tiempo_{k-1}).\,(aceleración_k + aceleración_{k-1})$$

Una vez calculada la velocidad, realizamos la misma operación para calcular la posición en dicho instante k:

$$posición_k = posición_{k-1} + 0.5\,.\,(tiempo_k - tiempo_{k-1}).\,(velocidad_k + velocidad_{k-1})$$

La implementación en el microcontrolador de estas funciones se puede encontrar en el código de implementación situado en anexo 5.

Al estar utilizando sensores inerciales, introducen ruido que puede confundirse con un incremento de aceleración o velocidad, por lo que es un gran problema ya que realizaríamos una o incluso dos veces la integración de ese dato erróneo, lo que llevaría a una variación de la posición que no existe.

Para ello, implementaremos una puerta de ruido tras el cálculo de la aceleración total, que explicaremos más en profundidad después. De esta manera, conseguimos eliminar una gran parte de componentes provenientes de ruido en estado de reposo.

Como finalización de este punto, debemos realizar una reflexión acerca de la primera inicialización de las componentes $a_{k-1}$ y $v_{k-1}$

Puesto que el movimiento que vamos a realizar se va a efectuar de una forma manual y el acelerómetro posee gran sensibilidad, es importante realizar una distinción entre velocidad nula y aceleración nula.

Cuando la aceleración es nula en un instante k, podríamos dudar de si proviene de derivar una velocidad constante o si de verdad la velocidad también es nula. Si nos ponemos en el contexto del movimiento que vamos a realizar, es muy complicado que se realice a una velocidad constante, por lo que consideraremos que no existe dicha variable constante, esto se traduce a que, para una velocidad nula, el incremento de la posición es cero.

Tras las pruebas realizadas sobre un solo eje, podemos concluir que cuando existe una aceleración nula, esto es cuando realizamos la calibración inicial, la velocidad se encuentra en valores próximos a cero, lo que indica una variación en la posición también nula.

## 3. Origen de los errores

### 3.1. Error debido a la integración numérica

El cálculo y solución de integrales definidas mediante métodos de integración numérica son siempre valores aproximados.

El método de la regla del trapecio posee un error E, de valor [15]:

$$E_t = -\frac{1}{12} f''(\xi)(b-a)^3$$

Donde $\xi$ corresponde a un número perteneciente al intervalo [a,b].

Las mediciones por integración nunca son una buena idea, ya que acumulan errores de ruido y medición, lo que provoca con el tiempo una mayor deriva de los datos.

Dado que estamos integrando entre instantes de tiempo muy pequeños, pero realizamos la integración de un gran número de muestras, este error será de un orden muy bajo al principio, pero que siempre será arrastrado y al poco tiempo será crítico.

### 3.2. Error de los componentes inerciales

La fuente de error principal de un acelerómetro es el "*bias*" ($m/s^2$) (offset de la señal de salida sobre el valor real), el cual difiere de cada acelerómetro concreto (en el siguiente punto analizamos el sensor utilizado). Este error es constante con valor ε, el cual para calcular la posición debemos integrar en dos ocasiones, lo que se traduce en un error cuadrático a lo largo del tiempo en la posición estimada, con valor:

$$p(t) = \varepsilon \times \frac{t^2}{2}$$

En el giroscopio, al responder a vibraciones, a medio y largo plazo tiene deriva, aunque el sensor esté estático. También, al igual que antes, la fuente de error principal es el "bias", donde en este caso el crecimiento es lineal a lo largo del tiempo de la forma:

$$\theta(t) = \varepsilon * t$$

Por ello, en el apartado cuatro explicamos soluciones para corregir lo máximo posible estos errores y la importancia de realizar una correcta calibración de todos los sensores a emplear y realizar re-inicializaciones periódicas siempre que sea posible para que el error sea mínimo. De esta forma, se convierten en una buena alternativa para complementar a otros sistemas de posicionamiento.

## 3.3. Error del sensor

El giroscopio que integra el sensor utilizado posee un error del 2% y podemos tener ±0,05 dps/˚C por lo que es importante trabajar a una temperatura adecuada ya que cada grado que aumente/disminuya en el sensor nos afecta en el error.

En la situación de que por un cable circule una intensidad superior a 10mA, los datos del magnetómetro/brújula pueden ser enviados con un error proporcional a la corriente, por ello es recomendable separar unos milímetros los cables del sensor.

Analicemos la sensibilidad de nuestros sensores, medida en función del bit menos significativo *(Least Significant Bit)*:

- Acelerómetro: 16384.000000 LSB/g
- Giroscopio: 131.072006 LSB/deg/s
- Magnetómetro: 0.776487 LSB/G

De nuevo, se pueden encontrar más datos de interés acerca de la precisión en las mediciones en el datasheet del dispositivo situado en el anexo 1.

## 3.4. Error de inclinación

Un error que afecta de manera considerable en nuestro sistema es el de la inclinación del sensor. En la posición ideal, cuando tenemos el brazo completamente estirado y con el sensor paralelo al suelo, la fuerza de la gravedad debería actuar en su totalidad sobre el eje z del acelerómetro, dejando las componentes restantes cercanas a cero.

Este error puede conllevar a un mal cálculo del ángulo de offset con el que inicializamos la descomposición de la componente gravitacional, donde un mínimo desvío angular puede llevar fácilmente al aumento de 1g en la fuerza de la componente gravitacional.

### 3.5. Error debido a un mal cálculo de la componente gravitacional

Para realizar la descomposición de la gravedad, se utiliza un filtro combinado de los ángulos provenientes de la aceleración y la velocidad angular.

Al estar utilizando fórmulas con senos y cosenos ya estamos obteniendo resultados con un cierto sesgo, que se convierte en un error fatal si estamos introduciendo ángulos imprecisos.

### 3.6. Error de muestreo de la señal

Debemos respetar las condiciones del teorema de Nyquist-Shannon, el cual establece que la frecuencia de muestreo de una señal debe de ser, al menos, el doble de la banda frecuencial que lleve la señal. El chip del microcontrolador Arduino Nano utilizado tarda 25 ciclos de reloj en realizar la conversión A/D.

En el algoritmo utilizado leemos muestras de aceleración y velocidad angular en los tres ejes, lo que supone que el tiempo entre dos muestras del mismo canal es 6x25 = 150 ciclos de reloj. Como se ha mencionado en la descripción del microcontrolador, su reloj trabaja a una frecuencia de 16MHz, por lo que podemos emplear un período máximo de muestreo de:

$$\frac{6\ x\ 25}{16MHz} = 9{,}73\mu s\ (106KHz)$$

Esta suposición es la más óptima y realista, ya que debemos tener en cuenta todas las declaraciones de variables, cálculos, llamadas al sistema, etc. En la simulación del sistema, podemos observar que la diferencia de tiempo entre dos muestras consecutivas del mismo eje son 5ms (200Hz).

Consecuentemente, podemos muestrear la señal analógica de entrada a una frecuencia máxima de 200Hz, por tanto, la máxima frecuencia que aceptaremos en la aceleración y velocidad angular es de:

$$\frac{f_{muestreo}}{2} = \frac{200Hz}{2} = 100Hz$$

En el momento de que la señal supere esta frecuencia superior, la señal original será distorsionada y en el proceso de muestreo y no podremos asegurar el teorema de Nyquist.

### 3.7. Offset de la velocidad tras el frenado

Cuando realizamos un movimiento partiendo del reposo, se produce un incremento en la aceleración, donde al finalizar el movimiento también está presente una aceleración de frenado de signo opuesto. Llevándolo al ámbito de la velocidad: es muy difícil generar manualmente una velocidad de frenado que tras medirla sea idéntica a la inicial del movimiento, por lo que, tras integrar, la diferencia entre áreas causará un dato diferente a cero, que deriva en la existencia de un offset.

Esto provoca que, al volver el sistema a un estado de reposo, la velocidad no regrese a cero y se mantenga una velocidad constante que en realidad no existe.

### 3.8. Error en el cálculo del desplazamiento

El objetivo final es poder calcular la posición final, por lo que esta variación de signo en la velocidad, provoca una mala resolución en la posición calculada, realizando un efecto de retroceso. Esto se explica porque los últimos datos integrados para obtener la posición son de signo opuesto a los que de verdad producen el movimiento.

### 3.9. Error de la resolución del microcontrolador

Nuestro microcontrolador utiliza un conversor analógico digital para convertir la señal analógica en un valor binario, es decir, transformar las señales analógicas a digitales (0 y 1). Para ello, se establece una resolución, que determina la precisión con la que se reproduce la señal original.

Conociendo el valor del voltaje máximo de entrada y la cantidad de bits que el dispositivo emplea para representar los datos en binario, podemos realizar un cálculo de la resolución del microcontrolador.

Arduino Nano utiliza 8 canales, con 10 bits a enviar al conversor digital. Esto significa que se mapean los pulsos de voltaje entre 0-5V a valores de enteros entre 0 y 1023. Esto nos da una resolución de 5V / 1024 unidades (0,0049V por unidad).

En nuestro caso estamos manejando tipos de dato *float,* donde la resolución tiene que ser finita y el error siempre estará tras cada operación y aproximación.

Por tanto, podemos asumir que el error proporcionado por el microcontrolador al sistema es un error despreciable respecto a los anteriores mencionados.

## 4. Soluciones aplicadas

En este apartado se explican los métodos empleados para corregir muchos de los errores anteriormente descritos.

### 4.1. Calibración del sensor

Al estar utilizando un sistema inercial compuesto de acelerómetro, giroscopio y brújula, es necesario realizar una inicialización o calibración inicial en la que se indique o mida la posición inicial sobre la que vamos a aplicar el movimiento.

Es posible estimar el valor del error *bias* mediante una medida del valor medio de la salida del acelerómetro a largo plazo cuando no se experimenta ninguna aceleración. Con ello, se puede recalibrar la medida de aceleración periódicamente, de forma que el error se reduzca. Los errores de *bias* no corregidos son los que limitan, principalmente, el rendimiento del sistema inercial de localización.

En el caso de la velocidad angular sucede lo mismo, podemos estimar el valor del error bias mediante una medida del valor medio de la salida del giroscopio a largo plazo cuando no se experimenta movimiento. Las medidas de salida se corrigen simplemente realizando la diferencia con el valor medio.

En el establecimiento del punto donde comienza el ejercicio, se indicarán unos *offsets* para referenciar nuestro punto de partida. Desde que el sensor comienza a transmitir, asumimos la existencia de ruido y escasa precisión, por lo que las primeras muestras no las utilizamos. A continuación, calculamos una media de las muestras posteriores.

La implementación en el microcontrolador de la calibración completa se puede encontrar en el código Arduino situado en el anexo 6.

## 4.2. Puerta de ruido

Se trata de un procesador dinámico de señal diseñado para eliminar los ruidos en situaciones de reposo, además de ofrecer un efecto de barrido automático. La función de una puerta de ruido es cortar el paso de toda señal que no supere un umbral prefijado.

En nuestro sistema, los umbrales han sido calculados realizando una media de los datos estando el sistema en reposo. De esta manera evitamos integrar datos de ruido tanto en reposo como durante el movimiento que nos lleven a una situación de incremento en la velocidad y posición que no se está produciendo.

Aplicaremos puertas de ruido en dos situaciones:

- Después de la medición del ángulo de giro por el giroscopio
- Tras obtener la aceleración total sin componente gravitatoria

La implementación en el microcontrolador de la puerta de ruido se puede encontrar en el código Arduino situado en el anexo 7.

## 4.3. Ángulos de Yaw, Pitch & Roll

Es de trascendental importancia saber cómo está orientado el sensor, ya que en función de esto podemos analizar en qué proporción está actuando la aceleración de la gravedad en las mediciones. Por este motivo, el sensor se coloca con los ejes de la forma más común: eje X será para el ancho, Y para profundidad y Z para la vertical.

Volviendo a la situación inicial explicada antes, se puede asegurar que la gravedad actúa casi en su totalidad sobre el eje Z, por lo que en los ejes X e Y la aceleración será cercana a cero cuando el cuerpo esté en reposo. Es aquí donde vamos a encontrar la solución a los errores derivados de una inclinación del sensor, tanto en la calibración como durante el movimiento.

Llevándolo al ámbito de una rehabilitación, cuando se realiza el movimiento con una trayectoria adecuada, pero con un mal gesto, conseguiremos detectar dicho descuido.

Es entonces cuando entran en escena los ángulos de *yaw (rotación sobre el eje Z), pitch (rotación sobre eje Y)* y *roll (rotación sobre el eje X)*.

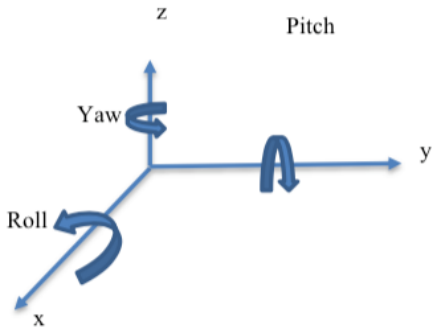



*Figura 15. Ángulos de Euler. Fuente: slideshare.net/rotacin-matricial Autor: Camilo Silva*

Con el acelerómetro y el cuerpo en reposo, sólo es posible calcular los ángulos de pitch y roll. Dado que estos ángulos se calculan a partir de la gravedad, el eje Z necesitaría otro sensor para poder medir su ángulo (*yaw*) [16].

Las fórmulas para el cálculo de estos ángulos son las siguientes:

$$Ángulo\ pitch = \text{atan}\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right)$$

$$Ángulo\ roll = \text{atan}\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

Así, podemos calcular la fuerza que ejerce la gravedad en cada eje:

$$g_x = 9.8 \times sen\ (pitch)$$

$$g_y = 9.8 \times sen\ (roll)$$

$$g_z = 9.8 \times cos\ (pitch) \times \cos(roll)$$

Restando estos valores a la aceleración que mide el sensor, obtenemos la aceleración en el sistema eliminando la componente gravitatoria.

Si realizamos un análisis de este método, válido en reposo, llegamos a la conclusión de que cuando tenemos el sistema en movimiento, los datos que mide el acelerómetro son el resultado de la suma de esta variación más la aceleración de la gravedad:

$$aceleración_{medida} = aceleración_{gravedad} + aceleración_{movimiento}$$

Volviendo a las fórmulas de *pitch* y *roll*, observamos que los ángulos calculados son dependientes de la aceleración, por lo que cuando adquirimos datos de aceleración debida a un incremento de la posición, estos son utilizados para calcular la *g* restada a la aceleración medida.

Por tanto, de esta manera, no podemos separar las aceleraciones, pero sí establecer una primera calibración para eliminar el efecto de la inclinación del sensor y realizar una representación de estos ángulos en un objeto para valorar la inclinación durante el movimiento.

En el apartado de pruebas realizadas se establece una demostración gráfica.

## 4.4. Giroscopio para obtener ángulos de inclinación

Nuestro sensor inercial posee un giroscopio de tres ejes que mide la velocidad angular del dispositivo en cada uno de los planos, por lo que vamos a obtener el ángulo resultante tras un incremento de tiempo.

$$\theta = \omega \,.\Delta t$$

Comprobamos que el giroscopio tiene mucho ruido, donde establecemos dos soluciones para tener ángulos más precisos. Estando en reposo, calculamos un offset a través del cálculo de la media de 800 muestras, el cual restaremos a cada muestra. Para situarlo aún más cercano a cero, aplicamos una puerta de ruido para las componentes de ruido no afecten.

## 4.5. Filtro combinado

Gracias a este filtro, vamos a obtener el ángulo definitivo para descomponer la fuerza gravitacional. La implementación combina un paso alto en el ángulo hallado anteriormente por el giroscopio, con un paso bajo para eliminar el ruido en el ángulo pitch/roll calculado con las aceleraciones del acelerómetro. Consideramos los signos y ángulos en función del cambio respecto a la referencia general y la del sensor [17].

$$\alpha_i = 0.98 * \left(\alpha_{i-1} - \theta_y\right) + 0.02 * pitch$$

$$\beta_i = 0.98 * (\beta_{i-1} + \theta_x) + 0.02 * roll$$

$$g_x = 9.8 \times sen\,(\alpha)$$

$$g_y = 9.8 \times sen\,(\beta)$$

$$g_z = 9.8 \times cos\,(\alpha) \times \cos(\beta)$$

Con este método conseguimos una buena estimación de la fuerza que ejerce la gravedad en cada eje en función de la orientación en la que se encuentre el sistema.

Tras esto, podemos eliminarla de las medidas recibidas por el acelerómetro y tener una aceleración que se corresponda con un incremento en la posición.

La implementación completa en el microcontrolador del filtro combinado se puede encontrar en el código Arduino situado en el anexo 8.

## 4.6. Utilización de un *threshold* para eliminar offset en la velocidad

Anteriormente vimos que tras integrar la velocidad se producía un offset fatal para el cálculo de la posición. Vamos a utilizar la aceleración para ver en qué situación se produce esa velocidad constante para hacer que vuelva a cero en situación de reposo.

Cuando estamos en reposo, la aceleración es cercana a cero, donde tras pasarla por la puerta de ruido provoca que su valor sea exactamente nulo. Esto nos hace pensar en una estructura que compare en todo momento el valor de la aceleración. Cuando ésta tenga un valor nulo, la velocidad deberá ser también nula. Para llevar la velocidad a cero de la manera más suavizada, la multiplicaremos por un factor entre 0 y 1 (*threshold*) [18], cuyo valor se ha calculado probando y observando con cual se obtiene un decaimiento más suave.

La implementación completa en el microcontrolador de la aplicación del threshold se puede encontrar en el código Arduino situado en el anexo 9.

## 4.7. Detectar cambio de signo en la velocidad

Para evitar realizar una integración de la componente de frenado de la velocidad, vamos a detectar un cambio de signo para saber cuándo comienza.

En el momento que detectamos este cambio de signo, iniciaremos un contador que esté un determinado número de muestras llevando la velocidad a cero.

Esto implica que durante el tiempo que dure el contador no realizaremos ningún movimiento, ya que se está llevando la velocidad a cero.

La implementación completa en el microcontrolador para realizar esta medida se puede encontrar en el código Arduino situado en el anexo 10.

En el apartado de *pruebas realizadas* podemos encontrar la demostración de cómo estos métodos consiguen solucionar muchos de los errores anteriormente mencionados.

# 5. Software desarrollado

## 5.1. Introducción

A continuación, se presenta el software y herramientas utilizadas para la implementación de este proyecto.

El sistema que utiliza el microcontrolador Arduino Nano se apoya en el software propio que nos facilita el fabricante, *Arduino Software* [19]. Para la representación del movimiento se ha utilizado el entorno de desarrollo *Unity 2017* [20] y para la generación del código y su compilación *Visual Studio 2018* [21].

En este apartado se establecen, mediante un diagrama de bloques, la estructura de funcionamiento que sigue nuestro sistema a nivel software en cada uno de los entornos mencionados.

## 5.2. Arduino

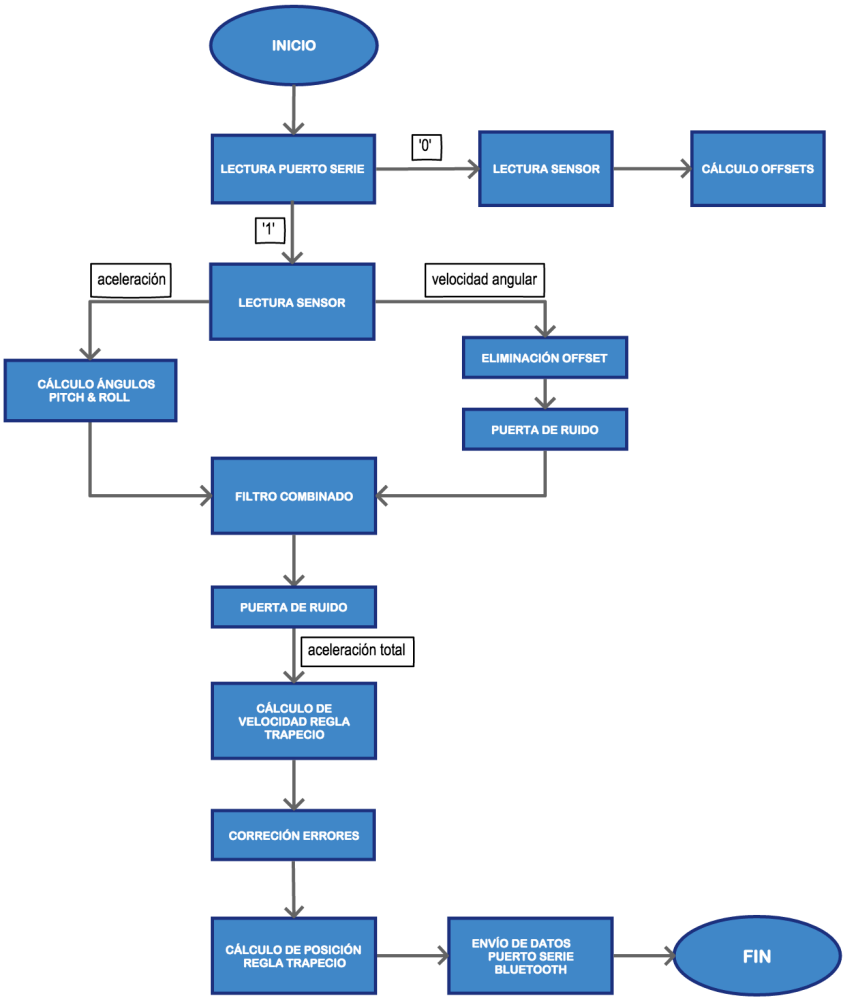



*Figura 16. Diagrama de flujo del software implementado en Arduino*

## 5.3. Sistema de control y representación en Unity

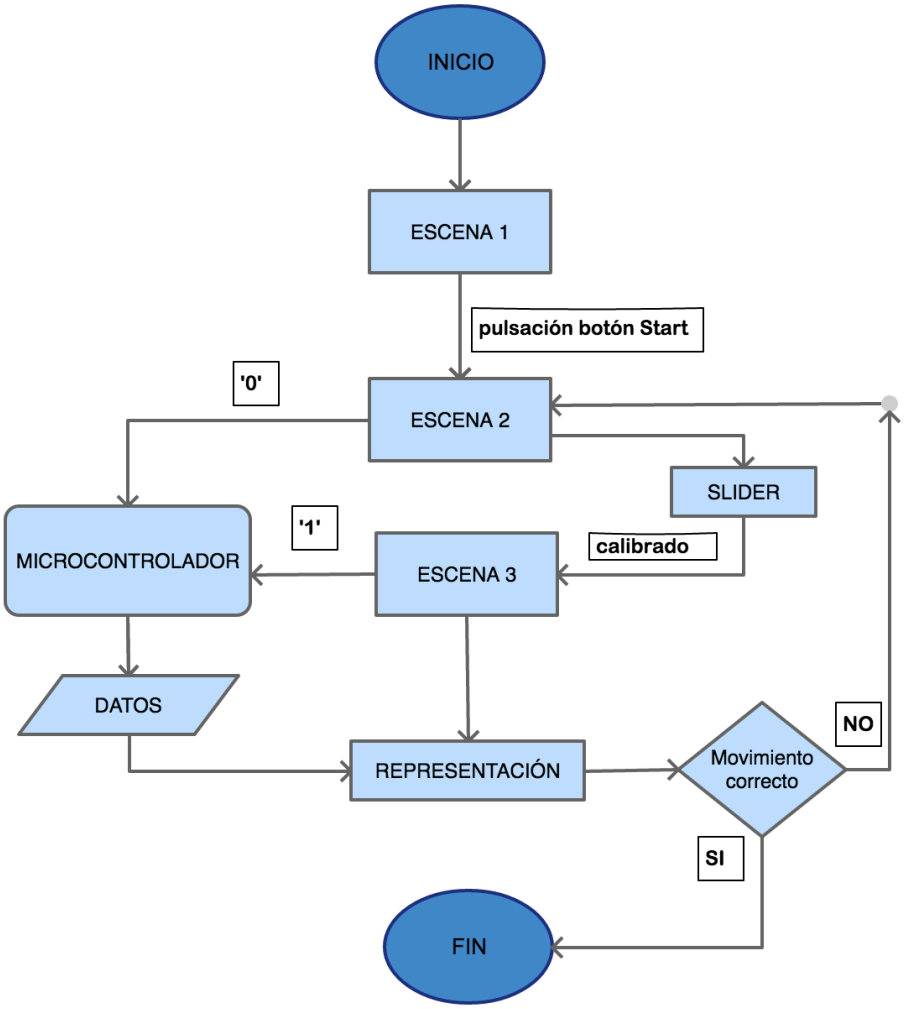



*Figura 17. Diagrama de flujo del software implementado en Unity*

# Capítulo 5. Entorno de visualización

Como hemos comentado anteriormente, utilizamos el software *Unity* para desarrollar un entorno que nos permita visualizar y realizar una valoración del movimiento realizado.

## 1. Escenas

El entorno se compone de un total de tres escenas:

### 1.1. Escena de comienzo

Se trata de un plano inicial donde tenemos un objeto de tipo *text* para indicar el título del proyecto y un objeto de tipo *button,* el cual indica el comienzo del ejercicio.

Comportamiento: al pulsar el botón de *Start,* cambiamos a la siguiente escena y abrimos el puerto serie para establecer la conexión con el microcontrolador.

### 1.2. Escena de calibración

Utilizamos de nuevo un objeto *text* para indicar que se está realizando la calibración y evitar moverse. Un objeto *slider* será el encargado de medir la duración de la calibración.

Comportamiento: tras entrar a esta escena, se envía un '0' al microcontrolador para indicarle que debe realizar el cálculo de los *offsets* necesarios. La barra de progreso indica el estado de la calibración. Cuando ésta acaba, se cambia automáticamente a la escena final.

### 1.3. Escena de representación

Esta escena estará dividida en dos partes diferenciadas:

La primera en la zona izquierda, donde observamos en la parte superior un objeto que simula un avión, el cual rotará acorde a los ángulos de *pitch* y *roll* que podremos visualizar en la parte inferior.

La zona central se compone de dos cámaras y un objeto de tipo *sphere*, donde la ventana principal se trata de un enfoque de frente a dicha esfera y en la parte superior derecha una vista desde arriba.

De esta manera, podemos visualizar el movimiento desde dos posiciones y realizar la valoración ayudándonos de la inclinación.

## 2. Representación de la rotación y movimiento

El software Unity posee un motor de físicas propio, donde un objeto, creado de la manera adecuada, puede estar sometido a fuerzas como la gravedad, entre otras. Además, permite detectar colisiones o caídas del objeto, de forma que el elemento se comporte de una forma muy realista.

En nuestro caso, para la rotación utilizamos la función *transform.Rotate*, donde le pasamos los parámetros de aceleración. De esta manera, el objeto que simula un avión realizará los movimientos en función de los ángulos de *pitch* y *roll*.

Para la representación del movimiento, el motor de físicas de Unity no lo vamos a utilizar, ya que estamos calculando una posición tras eliminar efectos de gravedad etc.
En este caso, con la función *transform.Position,* pasándole como parámetros los puntos en el espacio calculados en el microcontrolador y multiplicándolos por la escala adecuada, conseguimos el movimiento del objeto, en nuestro caso una esfera.

# Capítulo 6. Pruebas realizadas

## 1. Introducción

En este capítulo vamos a exponer los resultados obtenidos para ver claramente por qué es necesario utilizar el giroscopio para realizar la descomposición de la aceleración gravitacional en cada eje, así como la utilización del threshold y eliminación de la componente de frenado en la velocidad.

Para ello, efectuaremos medidas con los dos métodos de cálculo de la aceleración gravitacional, tanto en el eje Y como en el eje Z, hemos obviado el eje X por no extendernos más al tratarse de un caso similar.

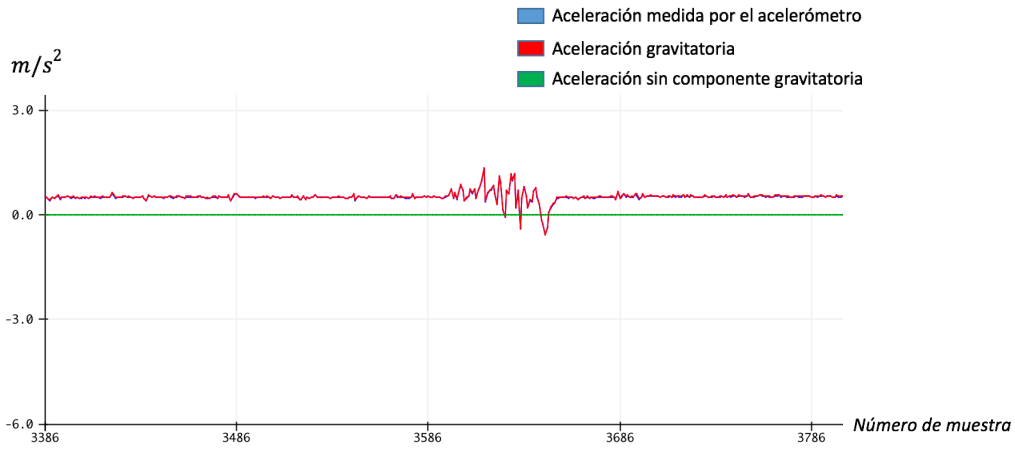
En el siguiente apartado realizaremos una representación de la aceleración medida por el acelerómetro (azul), el resultado del cálculo de la fuerza de la gravedad (rojo) y por último la diferencia de ambas (verde), que se tratará de la aceleración tras aplicar movimiento.

Posteriormente, en los apartados tres y cuatro, realizaremos movimientos para representar la velocidad y variación de la posición gráficamente en los tres ejes.

## 2. Pitch & Roll para eliminar componente gravitacional

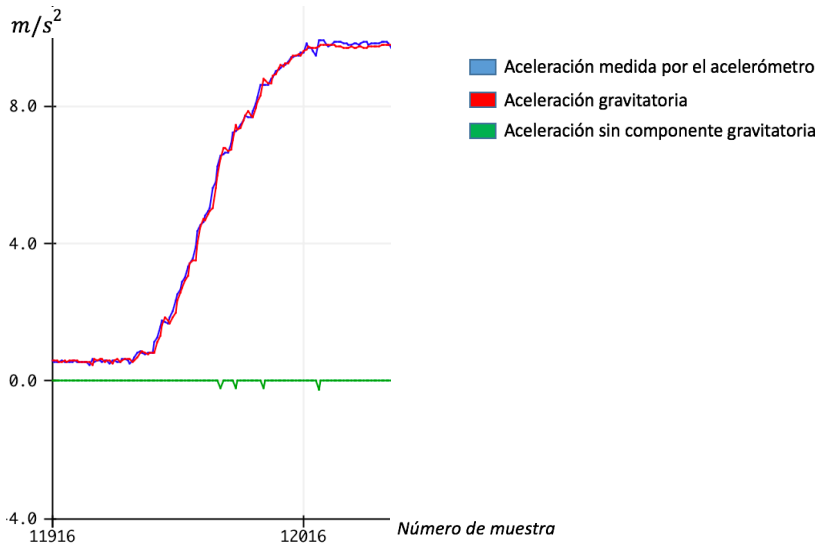
### 2.1. Análisis de la componente en el eje Y

Partiendo de la situación de reposo, realizamos un movimiento en el sentido positivo del eje Y. Podemos observar que como la aceleración es correcta, se consigue eliminar completamente el efecto de la gravedad en las medidas.




*Gráfica 1. Aceleración en el eje Y tras movimiento en dicho eje*

Ahora la medición se realiza cuando, desde la situación de reposo, rotamos sobre el eje x para acabar con el plano XY perpendicular al suelo.



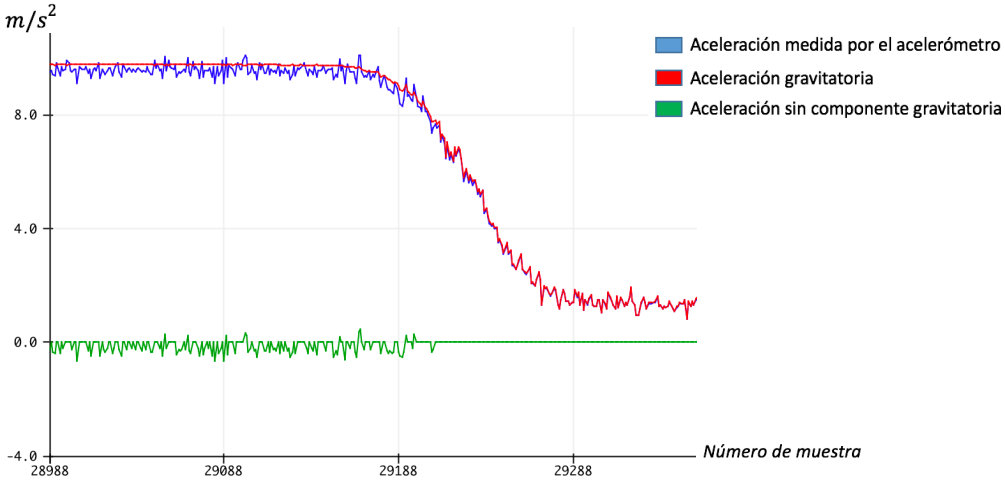

*Gráfica 2. Aceleración en el eje Y tras una rotación sobre el eje X*

Tal y como se observa, en ambos casos no existe una variación en la aceleración resultante, solamente unos picos que se trataría de ruido cuando supera el umbral de la puerta de ruido. Por tanto, en el objetivo de integrar la variación de movimiento, con este sistema no se obtienen datos concretos.

## 2.2. Análisis de la aceleración en el eje Z

Como se ve al comienzo de la gráfica, claramente la fuerza de la gravedad se sitúa sobre este eje, donde se consigue eliminar correctamente la componente de aceleración gravitacional, aunque con más problemas que en el caso anterior, ya que, si retomamos a la fórmula, se utilizan dos cosenos, lo que puede dar lugar a cierta imprecisión.

Posteriormente realizamos una rotación sobre el eje x, donde se ve un descenso considerable en la aceleración medida por el acelerómetro, compensada correctamente.




*Gráfica 3. Aceleración en el eje Z tras una rotación sobre el eje X*

Aquí se percibe de nuevo el problema, cuando se realiza el movimiento, se observa que la aceleración resultante (línea verde) no experimenta ninguna variación, que corrobora lo afirmado en apartados anteriores, el cálculo de $g_{x,y,z}$ no puede depender totalmente del ángulo calculado con los datos del acelerómetro.

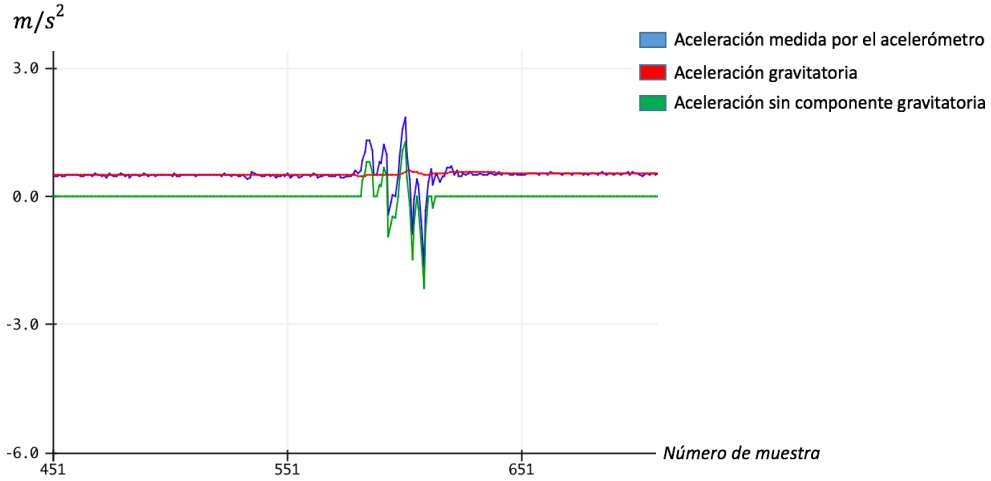
## 3. Giroscopio para eliminar componente gravitacional

En este apartado se analizan los resultados obtenidos tras utilizar el filtro combinado.

### 3.1. Análisis de la componente en el eje Y

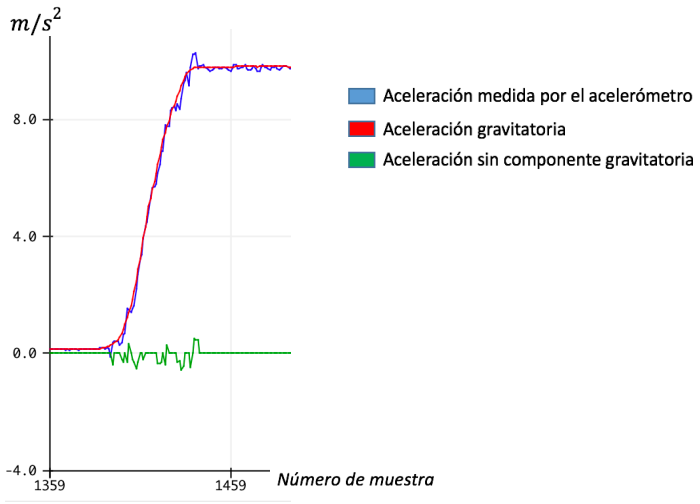
En reposo, la situación en el eje Y indica lo esperado: como la componente de la gravedad se sitúa en el eje Z, el eje Y no experimenta apenas esta fuerza. Aún así, el cálculo de *g* ayuda a situar la aceleración en un cero más preciso.

Tras esto, se realiza un movimiento en el sentido del eje analizado. Debido a que no se efectúa apenas variación en la inclinación, la gravedad sigue situándose en el eje Z, por lo que el valor de $g_y$ sigue siendo cercano a cero, y toda la aceleración que mide el acelerómetro se traduce en la aceleración provocada.




*Gráfica 4. Aceleración en el eje Y tras movimiento en dicho eje*

En la siguiente prueba, se rotará el sistema sobre el eje X hasta conseguir que el eje Y se sitúe perpendicular al suelo:




*Gráfica 5. Aceleración en el eje Y tras rotar sobre el eje X*

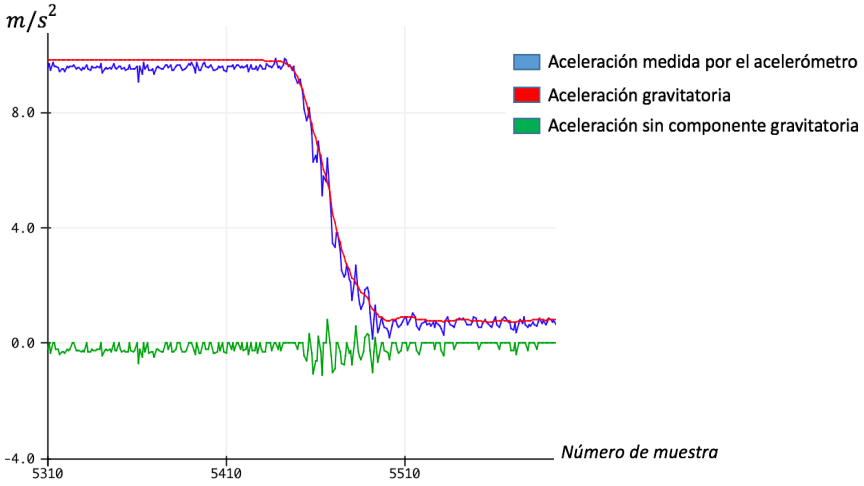
Como resultado se obtiene un incremento en la aceleración conforme el sistema se acerca a la posición final, donde la gravedad actúa prácticamente en su totalidad sobre este eje, contrastada correctamente por la componente $g_y$ tras volver a la situación de reposo.

Durante el movimiento se observa una variación de la aceleración final, eliminada con el método anterior y que ahora se puede utilizar.

## 3.2. Análisis de la componente Z

De nuevo, como en el anterior apartado, se aprecia en reposo que toda la fuerza de la gravedad actúa sobre el eje z, donde se percibe de nuevo una mala resolución en el cálculo de $g_z$, debido a los dos cosenos.

Ahora se realiza la rotación sobre el eje x para analizar las diferencias. Durante el movimiento se observa una variación en la aceleración resultante (verde) que se corresponderá a la inducida en el sistema. Tras realizar la rotación, vuelve al reposo, por lo que la fuerza de la gravedad se sitúa en el eje y, donde la aceleración resultante es cercana a cero.




*Gráfica 6. Aceleración en el eje Z tras rotar sobre el eje X*

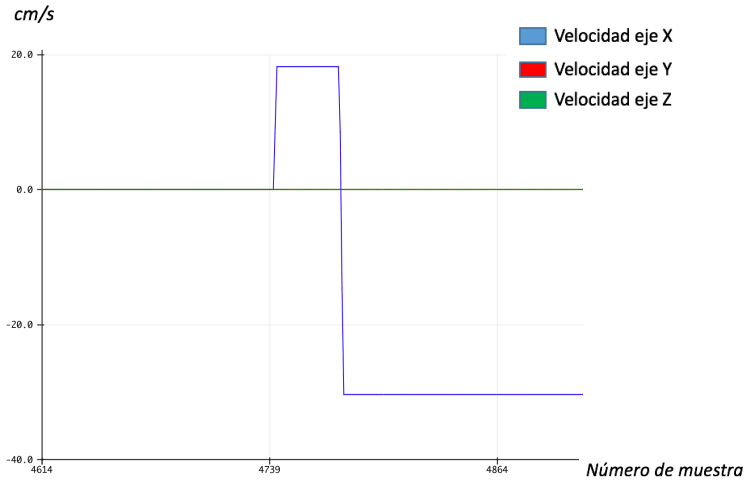
# 4. Cálculo de la velocidad

## 4.1. Offset no nulo de la velocidad tras la integración de la aceleración

**Caso 1: movimiento en sentido positivo del eje X.**

Como se observa, la velocidad aumenta en un primer momento, para después descender a valores negativos debido a que la aceleración de frenado es mayor y la diferencia tras la integración hace que la velocidad resultante posea signo opuesto.

Lo preocupante es que cuando el cuerpo vuelve al reposo, la velocidad se mantiene constante sin regresar a valores nulos, lo que arrastrará a integrar un valor de velocidad que se traduce a una variación de la posición estando el sistema estático.

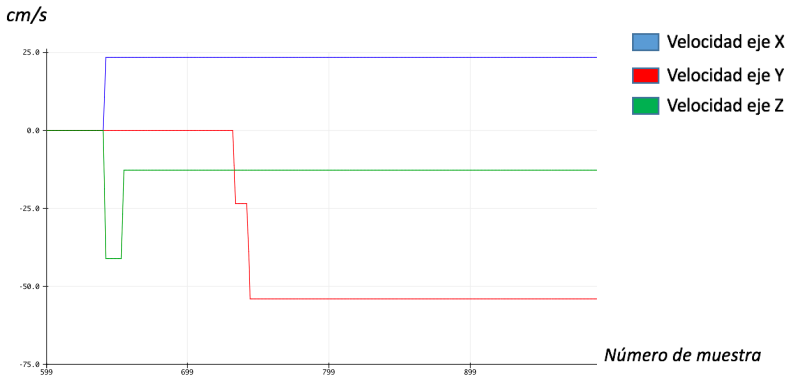



*Gráfica 7. Velocidad con offset no nulo en el eje X*

**Caso 2: movimiento en forma de arco.**

En esta situación, como la aceleración en el eje X es menor, la diferencia es positiva y la velocidad se mantiene sin cambiar de signo. En el eje Y ocurre lo mismo, salvo en dirección opuesta, donde el escalón que se percibe es debido a la puerta de ruido de la aceleración total. Mientas, en el eje Z, se observa una variación de la velocidad proveniente de lo comentado en el caso anterior: es muy difícil que las aceleraciones iniciales y de frenado sean iguales.

Independientemente de esto, la velocidad en los tres ejes se mantiene constante, sin hacer tentativa alguna de volver a cero cuando el movimiento termina.



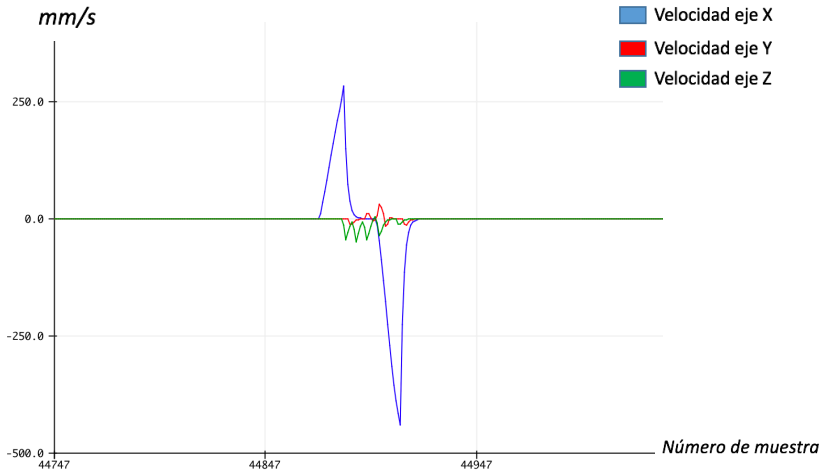

*Gráfica 8. Velocidad con offset no nulo en los tres ejes*

## 4.2. Corrección del error en la velocidad mediante *threshold*

**Caso 1: Movimiento en sentido positivo del eje X.**

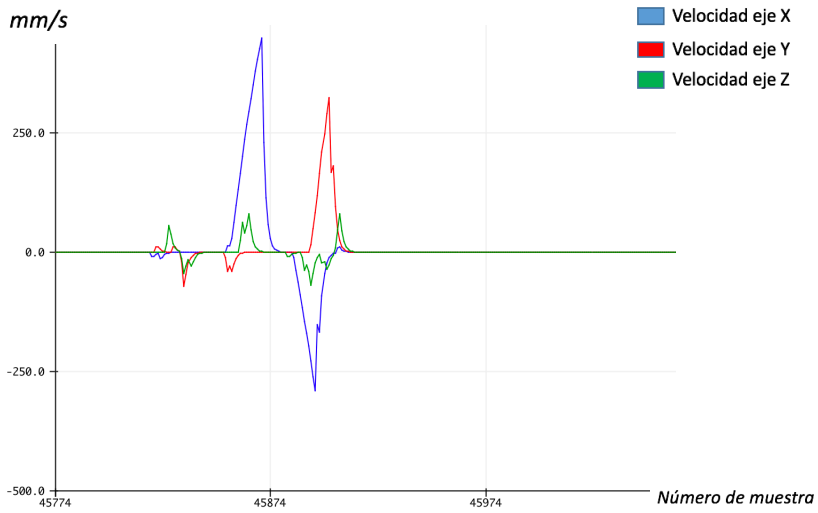
Tras corregir el problema anterior, la velocidad vuelve a posiciones nulas cuando el sistema está en reposo. Es ahora cuando se observa tras representar la posición (ver gráfica *13 en* página *46*) que primeramente se mueve en el sentido correcto (positivo) pero finalmente termina en una posición de signo contrario, lo que nos hace proporcionar unos datos erróneos de posición.

Esto se debe a que, como en la situación que teníamos con la aceleración, la diferencia resultante tras la integración de la velocidad no es nula, por lo que en la posición termina siendo un error fatal.




*Gráfica 9. Velocidad en eje X tras aplicar threshold*

**Caso 2: Movimiento en arco en ambos sentidos.**

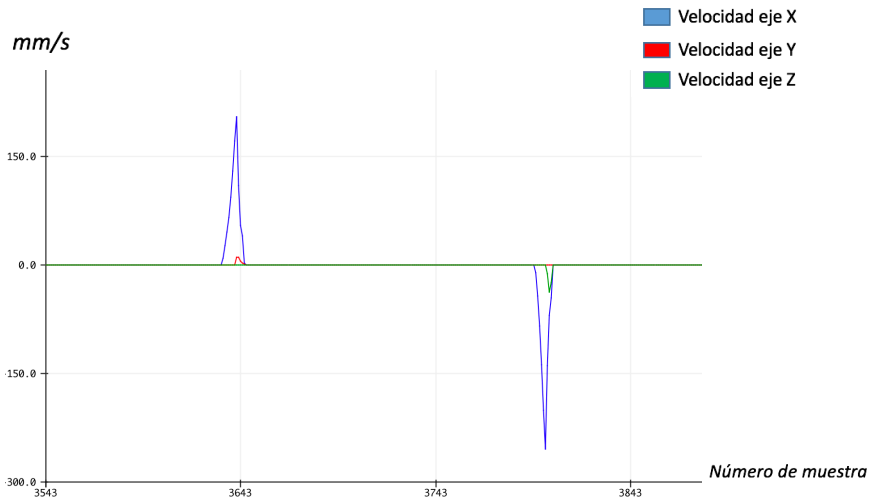



*Gráfica 10. Velocidad en los tres ejes tras aplicar threshold*

## 4.3. Eliminación de la velocidad de frenado

**Caso 1: Movimiento en sentido positivo del eje X y luego en el negativo.**

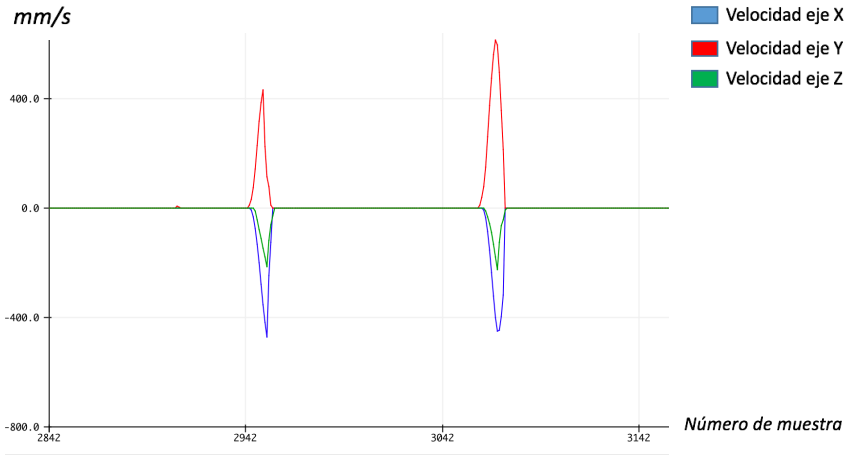
En este caso, tras realizar el movimiento se observa únicamente la velocidad provocada al sistema, sin tener en cuenta la componente de frenado.




*Gráfica 11. Velocidad en eje X tras eliminar velocidad de frenado*

**Caso 2: Movimiento en arco en ambos sentidos.**

En esta prueba se vuelve a garantizar que la componente de la velocidad de frenado no va a pasar por el lazo de integración.



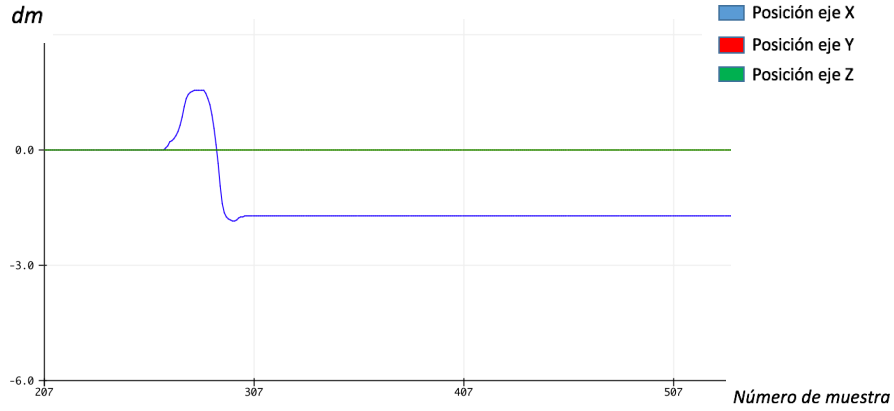

*Gráfica 12. Velocidad en los tres ejes tras suprimir la componente de frenado*

## 5. Cálculo de la posición

### 5.1. Posición con velocidad de frenado

**Caso 1: Movimiento eje X en sentido positivo.**

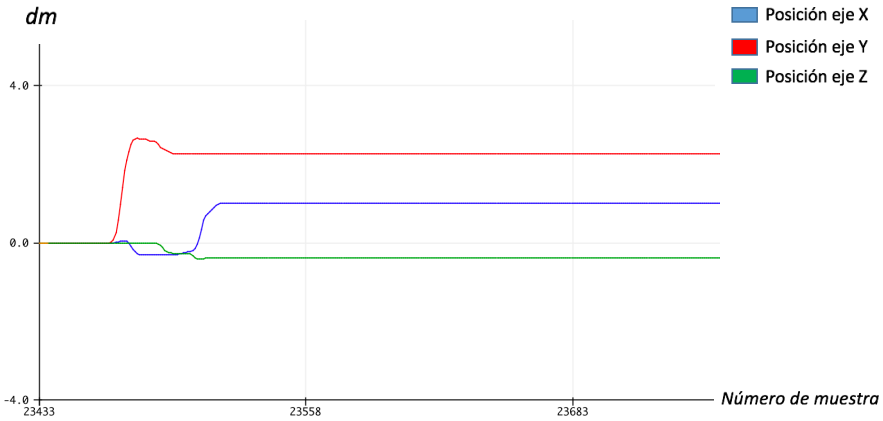
Se observa claramente el efecto de no eliminar la velocidad de frenado, donde la posición se realiza en sentido positivo y acaba siendo negativa.




*Gráfica 13. Posición en el eje X con velocidad de frenado*

**Caso 2: movimiento en forma de arco.**

Se distingue de nuevo que en los tres ejes se produce una variación en la posición, pero que finalmente realiza un cambio, provocado por la velocidad de frenado.

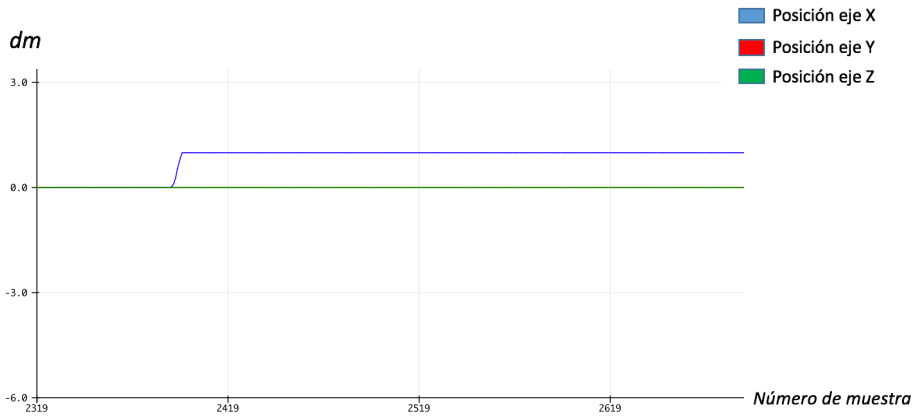



*Gráfica 14. Posición en los tres ejes con velocidad de frenado*

## 5.2. Posición sin la componente de frenado de la velocidad

**Caso 1: movimiento eje X en sentido positivo.**

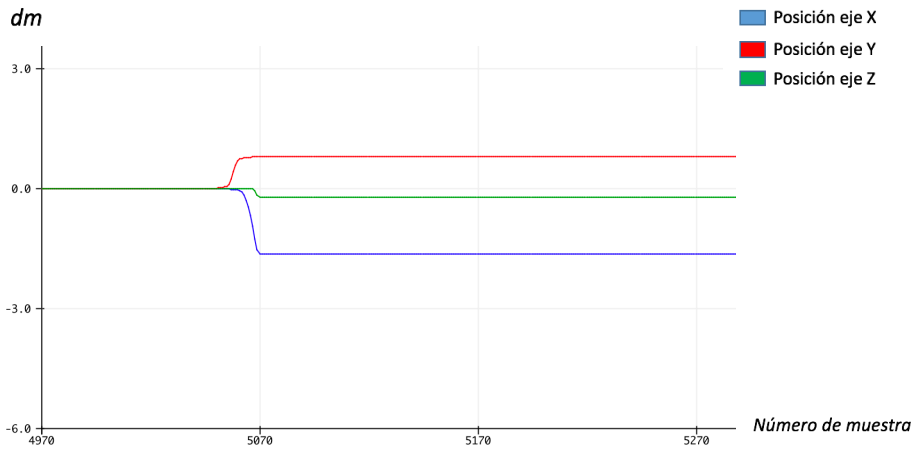
Tras eliminar la componente de frenado, al integrar la velocidad se obtiene una posición constante y sin variación.




*Gráfica 15. Posición en eje X tras eliminar la velocidad de frenado*

**Caso 2: Movimiento en forma de arco.**

En este caso también se mantiene la posición, sin realizar ningún tipo de variación.




*Gráfica 16. Posición en los tres ejes tras eliminar velocidades de frenado*

Las conclusiones que se obtienen tras este capítulo son claras, las cuales son fundamentales para realizar una valoración final del proyecto en el siguiente capítulo.

Se han representado todos los pasos hasta eliminar lo máximo posible las componentes que causan una deriva en la posición final.

Comenzando con la aceleración, es posible realizar varias veces el mismo movimiento obteniendo resultados parejos. Eso sí, tras haber realizado un entrenamiento previo y manejar a la perfección la técnica del movimiento.

En el objetivo de realizar un movimiento en el eje X una distancia de 10 centímetros, han sido necesarios un total de cinco intentos para lograr obtener una medición clara como demuestra la gráfica 15. Esto significa que, aunque el movimiento lo tengamos interiorizado y los primeros resultados de aceleración sean aceptables, posteriormente el sistema introduce y hace más grande el error.

Estas pruebas plantean un gran problema si el movimiento a realizar se ejecuta en los tres ejes y se debe valorar que todos los movimientos sean semejantes.

# Capítulo 7. Conclusiones y visión futura

## 1. Introducción

En este último capítulo se realiza una reflexión sobre los resultados obtenidos, analizando la precisión y exactitud de los mismos.

Estableceremos las principales conclusiones extraídas, así como las líneas futuras para mejorar el sistema y poder obtener resultados más precisos.

## 2. Conclusiones

El objetivo de este proyecto era realizar una captura y representación del movimiento mediante un sistema compuesto de sensores inerciales.

En base a las pruebas realizadas, se obtienen continuas imprecisiones. Para comenzar, recalcar que es muy complicado realizar movimientos perfectos, donde si tenemos una velocidad constante ésta no será medida. A esta carencia de exactitud en los movimientos mecánicos se suma que los sensores inerciales no logran capturar la variación de forma precisa.

De esta manera, vamos a tener siempre valores con una desviación o sesgo, que por muchos métodos que utilicemos para su suavizado o eliminación, se van a ir arrastrando.
Si a lo que se acaba de analizar, se añade que el método para calcular la posición a través de una aceleración es utilizando integrales numéricas, el error puede llegar a ser cuadrático.

Finalmente, se concluye afirmando que la utilización de un sistema compuesto únicamente de sensores inerciales no es capaz de realizar, de manera autónoma, un *tracking* efectivo en movimientos tan limitados como los de una rehabilitación.

Este sistema está muy lejos de su utilización a nivel profesional en el día a día, donde el método analizado durante todo el trabajo no es el adecuado para el objetivo propuesto.

## 3. Futuras líneas de actuación

Al hilo de las conclusiones extraídas en el punto anterior, con respecto a su utilización en la actualidad, este sistema está muy lejos de su utilización a nivel profesional en el día a día.

Las grandes vías de investigación y desarrollo están en la línea de realizar una detección de la posición y movimiento con un sistema inalámbrico, donde los sistemas inerciales se aplicarían en las zonas de baja señal.

Destacar que hoy en día el mundo de la tecnología está en un estado muy avanzado, donde gran parte de la población, como mínimo, dispone de uno de los siguientes dispositivos: *Smart watch, Smart band* o *Smartphone*.

Estos dispositivos incluyen tecnología inalámbrica como puede ser bluetooth o Wifi, además de acelerómetro y giroscopio, los cuales están muy optimizados e integrados en el hardware, por tanto, el reto estaría en poder acceder a los datos de los sensores inerciales para así evitar la parte de diseño hardware y realizar un sistema híbrido inalámbrico-inercial.

Centrándonos en las mejoras en el ámbito software de este proyecto, deberíamos:

- Realizar mejor precisión en los datos aplicando filtros más complejos que ayuden a eliminar al máximo el error.
- Establecer un mejor ajuste para las puertas de ruido utilizadas, aplicando más parámetros como el tiempo de ataque y decaimiento.
- La idea de utilizar un elevado número de sensores para entrenar el sistema y poder realizar una mecánica de movimientos acorde a los que puede realizar el cuerpo humano sería la mejor solución si sólo disponemos de sensores inerciales.
- Combinar los datos de los sensores con un sistema de posicionamiento inalámbrico, donde evitaríamos la calibración inicial situando la primera posición inicial a la última registrada por el sistema inalámbrico.
- En la parte de representación gráfica, el software Unity nos proporciona un potente entorno de desarrollo donde las mejoras, acompañadas de actualizaciones como mínimo anuales de Unity, siempre serán posibles.
- Realizar un registro de actividad dentro de la aplicación, donde el fisioterapeuta pueda ver a distancia si el paciente ha realizado los ejercicios y poder avanzar al siguiente tratamiento.

Por tanto, la idea de realizar una aplicación que consista en una representación, un juego o similar, para permitir un análisis y valoración de ejercicios de rehabilitación sigue teniendo viabilidad, siempre que se combine con un sistema de detección de movimiento inalámbrico para tener la mayor precisión posible.

# Referencias

**[1] Integración numérica :** A.Doubova, F.Guillén González, "Un Curso de Cálculo Numérico: Interpolación, Aproximación, Integración y Resolución de Ecuaciones Diferenciales", Universidad de Sevilla, 2007.

**[2] Actitud de un móvil** (a fecha 18/09/2018)
http://www.manualvuelo.com/INS/INS26.html
https://es.wikipedia.org/wiki/Indicador_de_actitud

**[3] Técnicas de posicionamiento:** Cristina Regueiro Senderos, "Error en el posicionamiento indoor de dispositivos móviles", Universitat Oberta Catalunya, 2014.

**[4] Robert Abel:** Harris M.Lentz III, "Obituaries in the Performing Arts", 2001.

**[5] Técnicas de detección de movimiento:** Rafael Ormaechea Izquierdo, "Desarrollo de una aplicación de detección de movimiento basada en comparativa estructural de imágenes", Universidad de Valladolid, 2012.

**[6] Dinamómetro:** Instrumento utilizado para medir fuerzas o calcular el peso de un objeto. El dinamómetro tradicional fue inventado por Isaac Newton, el cual basa su funcionamiento en el estiramiento de un resorte que sigue la ley de Hooke en el rango de medición.
Fuente www.wikipedia.es a fecha 9/09/2018.

**[7] Giroscopio:** Alejandro Caballero, "Buscando siempre no perder la orientación", 2016. Blog online en www.factoriadeingenieros.com a fecha 18/09/2018.

**[8] Magnetómetro:** Blog online a fecha 19/09/2018
http://www.guiaspracticas.com/detectores-de-metales/magnetometros, 2013.

**[9] Open-source:** https://en.wikipedia.org/wiki/Open-source_software

**[10] Protocolos comunicación:** Nuño Valencia, "Comparación de protocolos para micros", 2018. Web en línea a fecha 8/09/2018
https://www.drouiz.com/blog/2018/06/25/uart-vs-spi-vs-i2c-diferencias-entre-protocolos/

**[11] Sistema master-slave:** Sistema que permite el control simultáneo de múltiples receptores interconectados una distancia máxima de 200m. Los esclavos siguen las órdenes del maestro.
https://en.wikipedia.org/wiki/Master/slave_(technology)

**[12] Bus I2C:** Eduardo J. Carletti, "Comunicación – Bus I2C", 2017.
(Web en línea a fecha 8/09/2018)
http://robots-argentina.com.ar/Comunicacion_busI2C.htm

**[13] Integración Arduino con sensor:**
"Tutorial de Arduino y MPU-6050", 2014. (Web en línea a fecha 16/09/2018)
https://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

"Cómo usar un acelerómetro en nuestros proyectos de Arduino", Luis Llamas, 2016. (Web en línea a fecha 16/09/2018)
https://www.luisllamas.es/como-usar-un-acelerometro-arduino/

Orlando, "MPU6050 Arduino, Acelerómetro y Giroscopio", 2014. (Web en línea a fecha 8/09/2018) https://hetpro-store.com/TUTORIALES/modulo-acelerometro-y-giroscopio-mpu6050-i2c-twi/

**[14] Configuración Unity con microcontrolador:** Alan Zucconi, "How to integrate Arduino with Unity", 2015.
(Web en línea a fecha 8/09/2018)
https://www.alanzucconi.com/2015/10/07/how-to-integrate-arduino-with-unity/

**[15] Regla del trapecio:** David Vara Rodríguez, Juan Carlos Fraile Marinero, "Sistemas para determinar la posición y orientación de herramientas quirúrgicas en operaciones de cirugía laparoscópica". Universidad de Valladolid, 2014.

**[16] Ejes de rotación:** "Tilt Sensing Using a Three-Axis Accelerometer", Mark Pedley, 2013.
(Web en línea a fecha 16/09/2018)
https://www.nxp.com/files-static/sensors/doc/app_note/AN3461.pdf

**[17]** María José Pardilla Márquez, Ignacio Bosch Roig, Jorge Gosálbez Castillo, "Estudio de la aceleración y posición en sistemas inerciales por medio de Arduino". Universidad Politécnica de Valencia, 2016.

**[18]** Gerard Llorach, Alun Evans, Javi Agenjo, Josep Blat, "Position estimation with a low-cost inertial measurement unit", Universitat Pompeu Fabra, Barcelona, Spain.

**[19] Arduino Software:** Web oficial de Arduino a fecha 19/09/2018. https://www.arduino.cc/
José Enrique Crespo, "Aprendiendo Arduino"
(Web en línea a fecha 8/09/2018)
https://aprendiendoarduino.wordpress.com/

**[20] Unity:** Web oficial de Unity a fecha 19/09/2018. https://unity3d.com/

**[21] Visual Studio:** Web oficial Visual Studio a fecha 19/09/2018
https://visualstudio.microsoft.com/

# Anexos

## Anexo 1. Datasheet sensor inercial LSM9DS

https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf (a fecha 14/09/2018)

## Anexo 2. Datasheet microcontrolador Arduino Nano

https://store.arduino.cc/arduino-nano (a fecha 14/09/2018)

## Anexo 3. Datasheet módulo bluetooth HC-06

http://silabs.org.ua/bc4/hc06.pdf (a fecha 14/09/2018)

## Anexo 4. Desarrollo del método del trapecio

La idea clave es la utilización de trapecios para obtener aproximaciones precisas del área bajo la función que describe el movimiento de nuestro dispositivo.

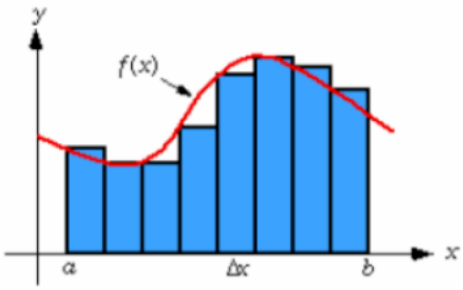



*Figura 18. Aproximación mediante la regla del trapecio*

Para ello, es necesario utilizar un polinomio de primer orden:

$$P_1(x) = f(a) + \frac{f(a) - f(b)}{b - a}(x - a)$$

Es entonces cuando al sustituirlo en la integral tenemos:

$$A = \int_a^b \left[ f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \right] dx$$

Tras resolver esta integral, queda como resultado:

$$\int_a^b f(x)dx \approx (b - a)\frac{f(a) + f(b)}{2}$$

## Anexo 5. Funciones de cálculo de velocidad y posición en Arduino

```
void getVelocity()
{
  for (int i = 0; i < 3; i++)
  {
    vel[i] = vel_1[i] + 0.5f * (acc[i] + acc_1[i]) * (incrementoT / 1000.0f);
  }
}

void getPosition ()
{
  for (int i = 0; i < 3; i++)
  {
    pos[i] = pos_1[i] + 0.5f * (vel[i] + vel_1 [i]) * (incrementoT / 1000.0f);

    //actualizamos variables
    acc_1[i] = acc[i];
    vel_1[i] = vel[i];
    pos_1[i] = pos[i];
  }
}
```

## Anexo 6. Implementación de la calibración del sensor en Arduino

En este anexo se calculan los offset de aceleración y velocidad angular. Tras recibir el *flag=0* del software de representación, las primeras diez muestras leídas se desprecian. Posteriormente realizamos la media ponderada de las siguientes 800 muestras para obtener valores precisos. A partir del offset de aceleración, calculamos también los ángulos de *pitch* y *roll* para tener una inicialización de los ángulos.

```
if (flag == 0)
{
//Siempre comenzamos la calibración con los buffer vacíos
    buff_acc_x = 0; buff_acc_y = 0; buff_acc_z = 0
    buff_gyro_x = 0; buff_gyro_y = 0; buff_moduloG = 0;


//810 lecturas, 10 primeras se desprecian y se utilizan las 800 siguientes
    for (int i = 0; i <= 10 + numero_lecturas; i++)
    {
      sensors_event_t accel, mag, gyro, temp;
      lsm.getEvent(&accel, &mag, &gyro, &temp);

      //Quitamos las primeras 10 muestras
      if (i > 9 && i < (numero_lecturas + 11)){
             acc_x = accel.acceleration.x;
             acc_y = accel.acceleration.y;
             acc_z = accel.acceleration.z;

        buff_acc_x = buff_acc_x + acc_x;
        buff_acc_y = buff_acc_y + acc_y;
        buff_acc_z = buff_acc_z + acc_z;

        gyro_x = gyro.gyro.x;
        gyro_y = gyro.gyro.y;

        buff_gyro_x = buff_gyro_x + gyro_x;
        buff_gyro_y = buff_gyro_y + gyro_y;

      }//FIN 810 LECTURAS
```

```
      //Calculamos los offsets de aceleración y velocidad angular
      if (i == (numero_lecturas + 10))
      {
        acc_x_off = buff_acc_x / numero_lecturas;
        acc_y_off = buff_acc_y / numero_lecturas;
        acc_z_off = buff_acc_z / numero_lecturas;

        gyro_x_off = buff_gyro_x / numero_lecturas;
        gyro_y_off = buff_gyro_y / numero_lecturas;

        pitch_off = 180 * atan(acc_x_off / sqrt(acc_y_off * acc_y_off +
        acc_z_off * acc_z_off)) / M_PI ;
        roll_off = 180 * atan(acc_y_off / sqrt(acc_x_off * acc_x_off +
        acc_z_off * acc_z_off)) / M_PI ;

      }//fin cálculo offsets

    }//fin bucle 810 iteraciones, ponemos flag a 1 para no volver a entrar

      flag = 1;

  }//FIN CALIBRACIÓN
```

## Anexo 7. Implementación de una puerta de ruido en Arduino

Cálculo de la puerta de ruido para el giroscopio, que se aplica de la misma manera, con otros umbrales a la aceleración total calculada.

```
float puertaRuido = 1.0f; //umbral de 1 grado

if (abs(gyro_x) < puertaRuido)
    gyro_x = 0.0f;

if (abs(gyro_y) < puertaRuido)
    gyro_y = 0.0f;
```

## Anexo 8. Filtro combinado en Arduino

```
//Calculamos componentes temporales en milisegundos
tiempoMuestraActual = millis();
incrementoT = (tiempoMuestraActual - tiempoMuestraAnterior);

//Cálculo de los ángulos a partir de la velocidad angular del giroscopio
theta_x = (gyro_x) * (incrementoT) / 1000.0f;
theta_y = (gyro_y) * (incrementoT) / 1000.0f;

//Ángulos en función de la aceleración medida por el acelerómetro
pitch_Acc = atan(acc_x / sqrt(acc_y * acc_y + acc_z * acc_z));
roll_Acc = atan(acc_y / sqrt(acc_x * acc_x + acc_z * acc_z));

//Implementación del filtro combinado
pitch_Total = 0.98 * (pitch_Total_1 - theta_y) + 0.02 * pitch_Acc;
roll_Total = 0.98 * (roll_Total_1 + theta_x) + 0.02 * roll_Acc;

//Cálculo de la componente gravitacional en cada eje
g_x = 9.8 * sin (pitch_Total);
g_y = 9.8 * sin (roll_Total) ;
```

```
g_z = 9.8 * cos (roll_Total) * cos (pitch_Total);

//Obtentemos la aceleración sin la fuerza de la gravedad

acc_total_x = acc_x - g_x;
acc_total_y = acc_y - g_y;
acc_total_z = acc_z - g_z;

//Por último actualizar los resultados
roll_Total_1 = roll_Total;
pitch_Total_1 = pitch_Total;
tiempoMuestraAnterior = tiempoMuestraActual;
```

## Anexo 9. Threshold para eliminar offset de la velocidad en Arduino

```
float factor = 0.5f;

for (int k = 0; k < 3; k++)
  {
    if (acc[k] == 0.0f)
      vel[k] = vel[k] * factor;
  }
```

## Anexo 10. Eliminar velocidad de frenado en Arduino

```
for (int i = 0; i < 3; i++)
  {
    if ((vel[i]*vel_1[i]) < 0) //cambio de signo en la velocidad

      contador = 100; //100 muestras, a 200Hz de frecuencia de muestreo
                      //estaremos 0.5segundos sin realizar movimiento
  }

  if ( contador > 0 )
{
    vel[0] = 0.0f;
    vel[1] = 0.0f;
    vel[2] = 0.0f;
    contador--;
  }
```

## Anexo 11. Conexión puerto serie en C#

```
public SerialPort arduino;
public string portName;
//puerto por cable usb"/dev/tty.usbserial-A906P0GW"
//puerto bluetooth "/dev/tty.HC-06-DevB"

void Start()
{
  arduino = new SerialPort(portName, 9600);
  arduino.Open();
}
```

## Anexo 12. Envío y lectura de datos por puerto serie en C#

```
public Vector3 leerPosicion()
{
       //DEBEMOS PROTEGER QUE EL PUERTO ESTÉ ACTIVO

        if (arduino.IsOpen)
        {
            arduino.Write("1");//escritura en puerto serie
            dataString = arduino.ReadLine();//lectura del puerto serie

            Debug.Log("Posiciones: " + dataString);

            dataBlocks = dataString.Split(',');

            posiciones[0] = float.Parse(dataBlocks[0]);
            posiciones[1] = float.Parse(dataBlocks[1]);
            posiciones[2] = float.Parse(dataBlocks[2]);

            return posiciones;
        }


        else
         {
              Debug.Log("PUERTO NO ABIERTO...");
              return Vector3.zero;
         }
}
```